VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Umělá inteligence v počítačových hrách

Artificial Intelligence in Computer Games

2017 Jan Wlosok

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student:	**Bc. Jan Wlosok**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	2612T025 Informatika a výpočetní technika
Téma:	Umělá inteligence v počítačových hrách Artificial Intelligence in Computer Games
Jazyk vypracování:	čeština

Zásady pro vypracování:

S rostoucí rychlostí a možnostmi dnešních počítačů rostou i nároky na kvalitu a komplexnost jednotlivých částí interaktivních aplikací. Zatímco se vykreslování, efekty, animace posouvají dopředu velkou rychlostí, herní umělá inteligence se vyvíjí pomaleji. Přesto se najde pár výjimek, které ukazují, že sofistikovanější umělou inteligencí lze zážitek z interaktivní aplikace posunout na novou úroveň. Cílem této práce je zaměřit se na umělou inteligenci herních pomocníků a možnosti vylepšení oproti aktuálnímu stavu. Pro realizaci jednotlivých ukázek využijte Unreal enginu 4.

1. Nastudujte a popište aktuální možnosti herní umělé inteligence.
Shrněte výhody a nýhody aktuálních řešení.
2. Vyberte vhodné metody pro implementaci UI pomocníka a podrobně je popište.
3. Vytvořte ukázkové aplikace, které budou demonstrovat typy chování vytvořené UI. Pro implementaci vhodně zkombinujte C++ kódy a blueprint scripty.
4. Odůvodněte použití jednotlivých postupů a srovnejte výsledky jednotlivých metod.

Seznam doporučené odborné literatury:

[1] Mark Dave: Behavioral Mathematics for Game AI. 2009, ISBN-13: 978-1584506843
[2] Rabin Steven: Game AI Pro 2: Collected Wisdom of Game AI Professionals. 2015, ISBN 978-1482254792

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Němec, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Rád bych poděkoval vedoucímu práce Ing. Martinu Němcovi, Ph.D. za cenné rady při tvorbě této diplomové práce.

Abstrakt

Cílem této práce bylo nastudovat možnosti herní umělé inteligence, popsat části, ze kterých se skládá a naimplementovat ukázkovou aplikaci demonstrující popsané postupy. V první části práce byl popsán rozdíl mezi akademickou a herní umělou inteligencí. Dále práce obsahuje popis aktuálního stavu herní umělé inteligence a jednotlivých typů herní umělé inteligence. V další části se práce zabývá implementací systému vnímání a rozhodovací části umělé inteligence. V této práci byla také popsána tvorbu audio-vizuální stránky umělé inteligence. Pro demonstraci postupů popsaných v této práci byla vytvořená ukázková aplikace. Tato aplikace demonstruje chování umělé inteligence pomocníka a nepřátel.

Klíčová slova: Umělá inteligence, Behavior tree, Systém vnímání, Animace, Herní engine, Unreal engine

Abstract

The goal of this thesis was to study the possibilities of game artificial intelligence, describe the parts from which it is composed and implement an example application demonstrating the described procedures. The first part of the thesis describes the difference between academic and game artificial intelligence. The thesis also contains a description of the current state of the game artificial intelligence and individual types of game artificial intelligence. The next part of the thesis deals with the implementation of perception system and decision-making part of artificial intelligence. This thesis also describes creation of audio-visual part of artificial intelligence. To demonstrate the procedures described in this thesis, an example application was created. This application demonstrates the behavior of artificial intelligence of companion and enemies.

Key Words: Artificial intelligence, Behavior tree, Perception system, Animation, Game engine, Unreal engine

Obsah

Seznam použitých zkratek a symbolů **9**

Seznam obrázků **10**

1 Úvod **12**

2 Herní umělá inteligence **13**
2.1 Aktuální stav herní UI 13
2.2 Typy herní umělé inteligence 14

3 Struktura herní umělé inteligence **17**
3.1 Získání informací 17
3.2 Zpracování a vyhodnocení informací 18
3.3 Provedení akce 19

4 Zrak **20**
4.1 Zrak v Unreal enginu 20
4.2 Detekce objektů 20
4.3 Kontrola viditelnosti 22
4.4 Optimalizace 24

5 Sluch **25**
5.1 Implementace sluchu 26

6 Rozhodování **30**
6.1 Behavior tree 30
6.2 Rozhodování umělé inteligence nepřátel 31
6.3 Rozhodování umělé inteligence pomocníka 33

7 Pohyb UI **36**
7.1 Navigační systém 36
7.2 Pohyb umělé inteligence nepřátel 37
7.3 Pohyb umělé inteligence pomocníka 39

8 Animace **42**
8.1 Animation blueprint 42
8.2 Blendspace 43
8.3 Aim offset 44
8.4 Typy animací 44

8.5 Montage 45

9 Dialogový systém **46**
9.1 Rozdělení dialogů 46
9.2 Inicializace dialogů 48
9.3 Provedení dialogů 48

10 Implementace ukázkové aplikace **50**
10.1 Návrh ukázkové aplikace 50
10.2 Optimalizace ukázkové aplikace 51

11 Závěr **54**

Literatura **55**

Přílohy **56**

A Příloha na DVD **57**

Seznam použitých zkratek a symbolů

1D	–	Jednorozměrný
2D	–	Dvourozměrný
3D	–	Trojrozměrný
ABP	–	Animation blueprint
BP	–	Blueprint
LOD	–	Level of detail
NPC	–	Non-player character
RPG	–	Role Playing Game
UE	–	Unreal engine
UI	–	Umělá inteligence
XML	–	Extensible Markup Language

Seznam obrázků

1 Age of Empiers II [4] . . . 15
2 Diagram herní umělé inteligence . . . 17
3 Zrak ve hře Shadow Tactics: Blades of Shogun [8] . . . 18
4 Jednoduchá a komplexnější výseč. [10] . . . 21
5 Geometrie reprezentující primární(modrý) a periferní(fialový) zrak . . . 22
6 Viditelnost hráče [10] . . . 23
7 Vizualizace paprsků pro kontrolu viditelnosti . . . 24
8 Využití choke nodes pro sluch ve Splinter Cell: Blacklist [10] . . . 26
9 Využití navlinků pro propojení vyvýšených mist. . . . 27
10 Šíření zvuku. . . . 28
11 Diagram stavů UI nepřátel . . . 33
12 Vužití modulárního způsobu tvorby logiky UI. . . . 34
13 Vyhledávání úkrytu pomocí zón . . . 39
14 Nalezení nejbližšího bodu na cestě. . . . 41
15 Ukázka jednoduché state machine . . . 43
16 Blendspace - nastavení a editor . . . 44
17 Konverzace mezi dvěma NPC . . . 49
18 Ukázková aplikace . . . 51
19 Profiler - graf neoptimalizované aplikace . . . 52

Seznam výpisů zdrojového kódu

1 Ukázka dialogů s podmínkou. 46

1 Úvod

Vývoj her se mění velmi rychle. Každou chvíli je na trhu nový hardware, nová konzole, nová platforma či zobrazovací zařízení a je tak zapotřebí se neustále přizpůsobovat. Každý rok tak vidíme hry s detailnějšími modely a větším rozlišením textur. Objevují se stále komplexnější materiály a částicové systémy. Animace postav jsou rok od roku realističtější a i samotné postavy dosahují čím dál většího množství detailů. Herní světy jsou obrovské a stále se zvětšují. Lze tak říci, že z technického hlediska se hry vyvíjejí velmi rychle a tento vývoj je na první pohled viditelný. Jedna z částí her se ale nevyvíjí stejně rychle jako ty ostatní. Jedná se o umělou inteligenci ve hrách. Hráči i recenzenti mají pro umělou inteligenci ve hrách málokdy pochvalná slova.

Až na několik málo výjimek se za posledních 10 let herní umělá inteligence moc nezměnila. V některých případech je to úmyslné rozhodnutí vývojářů. V jiném případě to může být nedostatkem výpočetního výkonu, který je alokován pro jiné systémy. Ovšem ve většině případů se jedná o nedostatek prostředků a času věnovanému vývoji UI pro danou hru. Navíc UI není tou částí hry, která se zmiňuje v propagačních materiálech hry, a která by na hru nalákala další hráče. A tak je často vývoj UI zanedbán nebo jsou prostě použity zažité principy.

Tato práce se zabývá rozborem aktuálního stavu herní humanoidní UI a na praktických příkladech ukazuje jak je UI implementovaná v různých případech. Zároveň se v praktických příkladech ukazují možné změny.

2 Herní umělá inteligence

I přesto, že konkrétní definici se dá určit jen velmi obtížně, můžeme si přiblížit účel herní umělé inteligence. Je si třeba uvědomit, že herní UI a akademická UI jsou dva rozdílné odvětví umělé inteligence. V některých ohledech jsou si sice podobné, ovšem jejich cíl je velmi odlišný. Akademická UI se využívá např. v robotice, zpracování obrazu, jazykových procesorech, předpovědi počasí, atd. Jedná se o systém, který na základě získaných informací provádí operace pro maximalizaci úspěchu v činnosti, kterou provádí. Cílem herní UI je téměř pravý opak. Jedná se o část aplikace (hry), která dopomáhá k co nejlepšímu konečnému výsledku. Tím výsledkem je zábavný nebo zajímavý virtuální zážitek. To, jak herní UI funguje na pozadí, jestli realisticky simuluje zrak nebo sluch, nebo jestli podvádí a získává některé data systémové, není ve výsledku důležité. Důležité je, jaký dojem z UI má koncový uživatel. Dá se tak říci, že herní UI je pouze iluze či trik imitující reálie jak je známe. Pokud bychom měli použít přirovnání, akademická UI je jako člověk žijící v reálném světě, který získává informace o svém okolí a patřičně na ně reaguje. Ale herní UI je jako herec, který má svůj scénář, pohybuje se v nereálném světě, spoustu informací zná dopředu a aktivně reaguje pouze na některé informace.

Cílem herní UI je tedy dopomoct k co nejlepšímu interaktivnímu zážitku. Doplňuje tak ostatní částí jako je grafika, zvuk, dialogy, design atd. a sama o sobě je prakticky nepoužitelná. Zároveň je zapotřebí mít na paměti, že herní UI nesimuluje realitu, ale vytváří iluzi reality. A jelikož má k dispozici velmi málo výpočetního výkonu, často se používají různé zkratky a triky k dosažení požadovaného výsledku.

2.1 Aktuální stav herní UI

Pokud porovnáme UI z pohledu hráče, zjistíme, že za posledních několik let se UI moc nezměnila. Vývojáři sice UI vylepšují, ale tyto vylepšení jsou pro koncového uživatele většinou neviditelné. To, že se NPC dokážou pohybovat v daleko větších herních světech, že dokážou zvládat členitější terén nebo spolupracovat ve větších skupinkách, považují hráči za samozřejmost. Umělá inteligence tak dělá pořád to stejné, s pořád se stejným výsledkem, jen ve větším prostředí s lepším grafickým zpracováním. Můžeme porovnat například hry Mass Effect[1] a Mass effect: Andromeda[2]. Ty dělí deset let, během nichž byly vydány ještě další dva díly. Přesto je umělá inteligence nepřátel i pomocníků v obou titulech prakticky stejná. Nelze se tedy divit, že hráči i recenzenti si často na nedostatek inovací v umělé inteligenci stěžují.

Zároveň je ale třeba podotknout, že problémem není nedostatek realističnosti. Příliš realistické chování nepřátel totiž není zábavné. Takové chování se hodí do simulátorů, ovšem pro běžné hráče není atraktivní. O problémech, jaké mohou nastat při implementaci příliš chytré UI, se můžete dočíst v článku z roku 2014 publikovaném na portálu Kotaku[3]. Změna by tedy měla nastat v tom, jak se NPC prezentují hráči a jak se volby hráče promítají do chování NPC. Bohužel u nepřátel se nedostatku těchto detailů nemůžeme divit, jelikož jejich životnost v průběhu hry je velmi krátká a zdroje pro jejich vývoj velmi omezené.

V posledních několika letech se ale objevilo hned několik her, které herní UI posouvají dál a ukazují, že se vyplácí investovat čas a peníze do této části hry. Tyto hry mají dobře zvládnutou UI nepřátel, ovšem v čem excelují, je UI pomocníků. Zde je totiž obrovský prostor pro zlepšení, který sice ve většině případů funguje pouze jako kosmetická změna, ale výsledný zážitek ze hry dokáže velmi vylepšit.

2.2 Typy herní umělé inteligence

Stejně jako většina částí her, i herní UI se rozděluje na různé typy. To, jaký typ UI se ve výsledku použije, záleží na žánru výsledné hry.

2.2.1 Simulace a deskové hry

Tento typ herní umělé inteligence má nejblíže k akademické umělé inteligenci. Řeší se totiž problém, který má pevně daná pravidla a pevně dané hranice. U her jako jsou šachy nebo dáma totiž existuje konečný počet kombinací. Tento počet je samozřejmě obrovský, ale čistě teoreticky lze po každém tahu vypočítat všechny kombinace až do konce hry a vybrat ten optimální tah. Prakticky se vývojáři tohoto typu UI snaží vytvořit co nejrychlejší a nejpřesnější algoritmus pro vítězství v konkrétní hře.

2.2.2 Strategické hry

I přesto, že jsou to hry komplexnější, než klasické deskové hry, poměrně hodně z nich se některým deskovým hrám podobají. Asi každý může dosvědčit, že u hraní klasických strategických her, jako jsou Age of Empires 1 nebo Stronghold [5], hráč přemýšlí podobně, jako u hraní šachů. A podobně funguje i UI těchto her. Zde je samozřejmě obtížnější vypočítat konečný počet kombinací. Tyto hry totiž mají daleko větší hrací plochu, daleko více herních postav a navíc se odehrávají v reálném čase. K šachům by se lépe daly přirovnat tahové strategie jako např. Heroes of Might and Magic [6]. I přesto, že se umělá inteligence strategických her velmi podobá umělé inteligenci strategických deskových her, obsahuje jeden element navíc. Tímto elementem je nedokonalost UI. Část UI, která je ve většině typech herní UI velmi důležitá a dodává hráčům pocit reality.

2.2.3 Skriptovaná umělá inteligence

U skriptované UI by hodně lidí nesouhlasilo se zařazením mezi umělou inteligenci. Nedisponuje totiž prakticky žádnou inteligencí. Celý princip skriptované UI je v tom, že se jedná o NPC s předem určeným chováním. Pokud se ale podíváme do našeho popisu herní UI, zjistíme, že i tento typ tam spadá. Jedná se systém, který dopomáhá finálnímu interaktivnímu zážitku. Navíc i naskriptované postavy musí být schopny v určitých pasážích hry inteligentně provádět různé akce.

Obrázek 1: Age of Empiers II [4]

2.2.4 Umělá inteligence nepřátel

Toto je kategorie UI, která v posledních letech zaznamenala nejméně inovací. Ovšem důvodem není nezájem vývojářů, ale účel, za kterým je tento typ UI vytvářen. UI nepřátel je totiž v drtivé většině případů vytváří pro jediný účel, aby ji mohl hráč zneškodnit. Ve většině her, kde se tento typ UI využívá, je hráč tím silným jedincem a je v zájmu vývojářů aby se tak i cítil. Z toho důvodu se UI nepřátel nevytváří příliš inteligentní. Většinou se jedná o UI, které se umí pohybovat, krýt za překážkami a útočit (případně bránit). Samozřejmě jsou hry, kde je žádoucí, aby byli nepřátelé chytřejší. Většinou se ale vytváří poměrně přímočaré UI.

2.2.5 Umělá inteligence pomocníků

Tato kategorie UI je pravým opakem UI nepřátel. Kromě toho, že se jedná o NPC, které se snaží hráči pomoct, se jedná o systémově mnohem komplexnější UI. Hranice chytrosti UI jsou zde omezeny pouze výkonem současného hardware. Tato UI má ovšem daleko menší využití, než předchozí typ. Používá se totiž skoro výhradně ve hrách pro jednoho hráče, často s velkým důrazem na příběh. A právě věrohodný UI pomocník má zprostředkovat ještě intenzivnější zážitek ze hry. Zde se zároveň jedná o perfektní příklad herní UI, kde se při vývoji vynakládá velké úsilí při tvorbě nedokonalostí. Ve většině případů se totiž jedná o humanoidní UI a se správně vytvořenými nedokonalostmi a neideálními akcemi získává UI na lidskosti. U vývoje tohoto typu UI ve většině případu úzce spolupracuje hned několik vývojářských oddělení. Jmenovitě se jedná o UI programátory, game designery a animátory.

2.2.6 Umělá inteligence pro řízení běhu hry

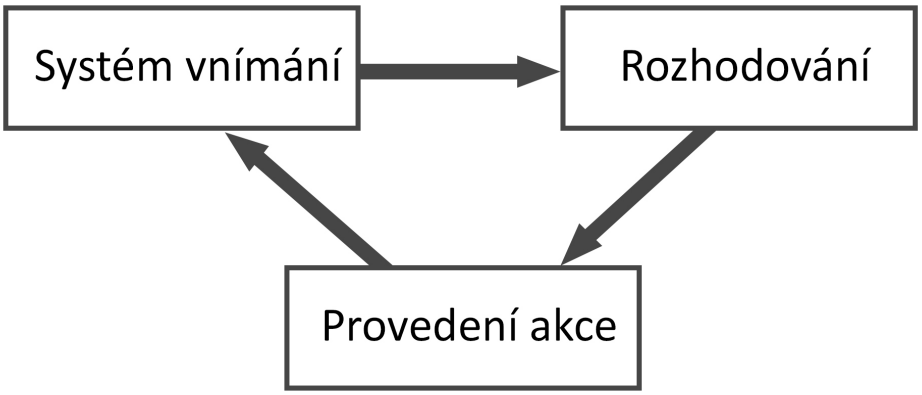
Speciálním typem umělé inteligence je systém pro řízení běhu hry. Většinou se jedná o systém, který dynamicky upravuje obtížnost úrovně. Analyzuje hráčův průchod úrovní a dokáže rozpoznat, zda má zvýšit či snížit obtížnost. Stejným způsobem se dá měnit i hratelnost. Tento systém se používá například v pasážích, kde je zapotřebí udržet hráče v napětí. Například dynamickým odebíráním zdrojů potřebných k průchodu úrovní na minimum. Speciálním případem této UI je systém, který uměle vytváří napětí. Používá se například v závodních hrách, kde se dynamicky pomáhá UI závodníkům tak, aby byli hráči pořád v patách. V roce 2008 byla vydána hra Left 4 Dead[7]. Právě tato hra využívala UI pro řízení hráčova zážitku. "The AI Director", jak nazvali tuto UI ve studiu Valve, byl zodpovědný za rozmístění nepřátel, dávkování akce a dokonce i vizuální a audio efekty.

2.2.7 Umělá inteligence pro procedurální generování obsahu

Zvláštním typem UI může být i systém pro procedurální generování obsahu. Může se jednat například o různé simulace tvorby terénu, které závisí na předchozích akcích hráče. Případně se může jednat o systém generující úroveň podle toho, jakým stylem hráč hraje. Zde je možná vidět podobnost s předchozím typem UI. Není neobvyklé kombinovat více takovýchto systémů pro získání ještě větší variability.

3 Struktura herní umělé inteligence

Komplexnost herní UI se liší podle typu hry, ve které je UI použita. Některé hry si vystačí s velmi jednoduchou UI, jiné budou využívat poměrně komplexních systémů. Hry mohou používat UI v podobě NPC nebo například systému pr ovládání jednotek. Bez ohledu na to, o jaký typ UI se jedná nebo jak složitý je výsledný UI systém, bude muset UI nejdříve získat data, na základě těchto dat rozhodnou co má dělat a poté tuto akci provést.



Obrázek 2: Diagram herní umělé inteligence

3.1 Získání informací

Nejdříve musí UI získat informace o světě. Můžou to být informace získané v rámci hratelnosti nebo například data, která UI agent získá pomocí svých smyslů. UI v akční hře bude spoléhat primárně na své smysly, zatímco UI ve strategických hrách bude na základě obtížnosti získávat menší či větší množství dat o herním světě systémově.

3.1.1 Systém vnímání

Systém vnímání je jeden ze způsobů, jakým NPC získává informace o světě. Ve většině případů se jedná o systém, který se snaží simulovat lidské smysly. Samozřejmě jak z výkonových tak i hratelnostních důvodů se nejedná o realistickou simulaci smyslů, nýbrž pouze o aproximaci. Často jsou navíc smysly NPC otupeny. Jedná se o způsob zlepšení zážitku hráče.

Z lidských smyslů se nejčastěji používá zrak 3 a sluch. Důvodem je samozřejmě to, že tyto dva smysly používáme i při hraní her. Hry nám neumožňují využít naše ostatní smysly. Nemůžeme se dotknout objektů ve hře ani je cítit nebo ochutnat. Navíc vzhledem k náplní většiny her a virtuálních aplikací, by stejně tyto smysly byly využity pouze velmi málo. A stejně tak jo my vnímáme herní svět, tak se stejným omezením vnímají herní svět i NPC.

Obrázek 3: Zrak ve hře Shadow Tactics: Blades of Shogun [8]

3.1.2 Ostatní podpůrné systémy

Často je daleko jednodušší využít pro získání nějaké informace jiný systém než zrak nebo sluch. Důvodem je fakt, že zrak ani sluch neimplementujeme realisticky. To se nám hodí při optimalizaci hratelnosti, ale zároveň to NPC v mnoha ohledech limituje. Proto některé informace, které bychom jako lidé získali pomocí zraku, dodáme NPC jiným způsobem.

3.2 Zpracování a vyhodnocení informací

Jakmile má UI potřebná data, musí se na jejich základě rozhodnout co dál dělat. Veškeré rozhodovací systémy potřebují nějaký vstup, aby mohly generovat výstup. Tím vstupem mohou být data, která UI získá za běhu hry, ale také data, která designer zadá při vývoji. Rozhodovací část, která se dá považovat za mozek UI, je pak zodpovědná za správné zpracování těchto dat. Implementace této části se liší podle využití UI. Ať už implementujeme rozhodovací část jakkoliv, po získání dat tato část rozhodne co má UI dělat.

3.2.1 Způsoby rozhodování

Existuje spoustu způsobů jakými na implementovat rozhodování UI. Můžeme použít konečných automatů, utility based system, hierarchical task network, goal oriented system, atd. Více informací o těchto systémech je možno najít v knihách Game AI Pro 1 a 2 [9][10]. V této práci budeme používat behavior tree (dále BT). Jedná se o jeden z nejpopulárnějších systémů pro

tvorbu logiky UI. Využívají ho enginy jako Unreal engine, Cry engine, ale například i Snowdrop engine.

3.3 Provedení akce

Jakmile se UI rozhodne co má dělat, je třeba tuto akci provést a nějakým způsobem své rozhodnutí prezentovat, aby hráč rozuměl, co UI dělá. Ve většině případů půjde o nějakou akci přímo ovlivňující hráče nebo herní svět. Mezi takové akci patří například pohyb po světě nebo útok na hráče. Neméně důležité jsou ale i akce čistě kosmetické. Může to být jednoduchý pohyb, animace či například dialog. Prezentace je velmi důležitou části UI, protože se pomocí ni dát navodit pocit velmi inteligentní UI i na základě jednoduchého rozhodnutí umělé inteligence. Navíc je nutné komunikovat stav UI zřetelně, aby hráč chápal co se děje a necítil se zmaten nebo ošizen. Je třeba si uvědomit, že nevytváříme hry a interaktivní aplikace pro prezentaci své UI, nýbrž pro vytvoření příjemného zážitku pro hráče.

4 Zrak

Jednou z klíčových komponent herní UI je zrak. Aby mohla UI reagovat na svět, ve kterém se nachází, musí nejdříve získat informace o jeho stavu. A právě největším zdrojem těchto informací je zrak. Samozřejmě se nejedná o simulaci reálného zraku. Jde spíše o aproximaci, jejíž výsledek se reálnému zraku blíží. Je totiž nutné, stejně jako u ostatních herních mechanismů, najít kompromis mezi reálností a zábavností.

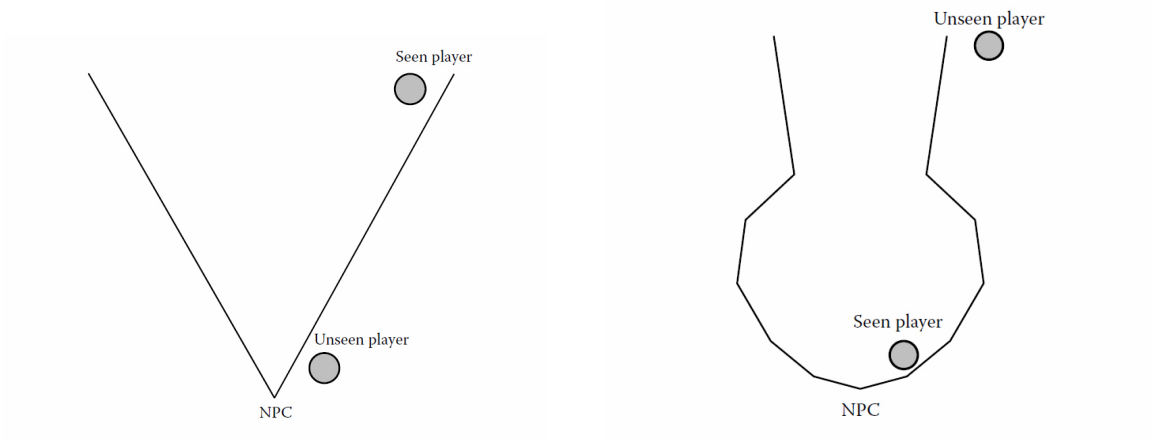
4.1 Zrak v Unreal enginu

Unreal engine disponuje vlastní implementací zraku. Ta je dostačující pro většinu jednodušších UI, ovšem pro naše účely je nevhodná. Jedním z největších nedostatků je nemožnost tento systém připojit na libovolnou část postavy. Zrakový systém UE je totiž použit u kontroléru postavy a tudíž můžeme maximálně nastavit dědění transformací z postavy. Pro naše účely ale potřebujeme zrakový systém připojit na hlavu postavy, aby se otáčel spolu s animacemi otáčení hlavy. Dalším nedostatkem je výseč, kterou zrakový systém UE používá. Ten totiž používá jednoduchou výseč a vzdálenost pro detekování objektů, což není ideální pro komplexnější UI. V naší implementací zrakového systému využíváme pro tento účel zrakovou geometrii. To nám dovoluje pečlivě vytvarovat zrakovou výseč.

Samozřejmě existuje možnost si aktuální zrakový systém upravit. Zdrojové kódy UE jsou dostupné a tak nám v tom nic nebrání. Bohužel po prozkoumání kódu narazíme na několik problémů. Perception komponenta je totiž v UE na implementovaná jako actor komponenta, což je komponenta neobsahující transformační funkcionalitu. Jinými slovy, nemá souřadnice a funkcionalitu s tím spojenou. Z toho vyplývá, že kromě námi požadovaných vylepšení bychom museli dopsat funkcionalitu scene komponenty. A i tak bychom narazili na problém v podobě nemožnosti použití dalších komponent v rámci zraku, jelikož není možné používat komponenty na jiné komponentě. Proto bylo v tomto případě lepší vytvořit vlastní řešení ušité na míru, místo abychom se snažili přizpůsobit si existující řešení.

4.2 Detekce objektů

U nejzákladnějšího zraku se viditelnost konkrétního objektu určuje splněním tří podmínek. První z nich je, zda se objekt nachází v dohledné vzdálenosti. U her se používají vzdálenosti dohledu daleko menší než v reálném světě, kde dokážeme rozeznat pohybující se objekt na stovky metrů. Je to designérské rozhodnutí (lepší hratelnost). Pokud je tedy objekt v dohledné vzdálenosti, musíme zjistit, zda je v adekvátní výseči. Tím eliminujeme například všechny objekty za UI. Nejjednodušším způsobem je použít skalární součin směru zraku a směru od UI k objektu. Poslední podmínka je viditelnost. Objekt totiž může být blízko a přímo před UI, ovšem může být schovaný za jiným objektem. Proto používáme raycasting pro detekování překážky mezi UI a sledovaným objektem. Pokud nenajdeme žádnou překážku, objekt je viditelný a UI může na



Obrázek 4: Jednoduchá a komplexnější výseč. [10]

tento fakt reagovat. Je třeba zmínit, že pořadí těchto podmínek je velmi důležité. Podmínky se snažíme řadit podle náročnosti provedení (od nejjednodušší) nebo podle četnosti neúspěchu.

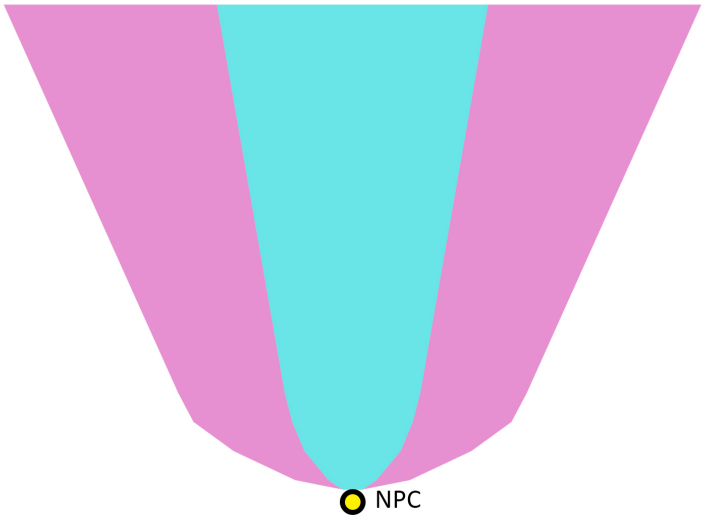
Jakýkoliv pokročilejší systém zraku staví právě na předchozích podmínkách. Vždy je třeba kontrolovat vzdálenost, úhel k pozorovateli a jeho viditelnost. Tyto podmínky se ale dají dále rozšířit. Nejzákladnějším rozšířením je přidání druhé výseče s větším úhlem pro detekování periferního vidění. Neméně častým rozšířením je rozdělení vzdálenosti na dvě části. Blízké objekty jsou rozpoznány ihned, zatímco objekty ve vzdálenější části jsou považovány pouze za podezřelé/neidentifikované. Ovšem i toto rozšíření se dá ještě vylepšit.

Jedním takovým vylepšením je úprava tvaru zrakové výseče. Je zapotřebí rozšířit zrak blízko UI, jinak může hráč nepozorovaně stát vedle UI na hranici jeho zorného pole. Například v The Last of Us (2013) [11] využili tohoto tvaru pro detekování prostředí 4.

Dále je také možné upravit tvar výseče v dálce, kdy do určité vzdálenosti se výseč rozšiřuje, ale poté se začne zužovat. Tímto způsobem můžeme simulovat snahu NPC zaostřit na vzdálený objekt. Samozřejmě je třeba volit řešení vhodné pro konkrétní typ hry.

V této práci byl vytvořen systém, jenž byl inspirován implementací zraku ve hrách Splinter Cell: Blacklist a The Last of Us. Obě hry jsou zaměřeny na plíživý postup hráče úrovní, a právě tento typ hry vyžaduje kvalitní implementaci smyslů. Zároveň obě tyto hry využívají místo klasické zrakové výseče geometrii 5. Tato geometrie je pak použitá místo kontroly vzdálenosti a výseče. Kontrola viditelnosti se pak provádí nad objekty uvnitř této geometrie.

Ve 3D editoru byla vytvořena geometrie reprezentující tvar zrakové výseče. I přesto, že je tento tvar inspirován reálným zrakem, v první řadě musíme dbát na hratelnost. Proto je tvar upraven pro naše potřeby tak, aby co nejlépe fungoval v našich ukázkách. U jiných typů her by se tento tvar mohl lišit. Tato geometrie nebude ve hře vykreslována, ovšem využijeme její kolize. Fyzické kolize u této geometrie vypneme, ale zapneme možnost detekování objektů nacházejících se uvnitř. Tato geometrie je následně připojená na model postavy NPC. Konkrétně na kost



Obrázek 5: Geometrie reprezentující primární(modrý) a periferní(fialový) zrak

reprezentující hlavu. Díky tomu se geometrie otáčí spolu s hlavou NPC a můžeme tak využívat animací k aktivnímu rozhlížení.

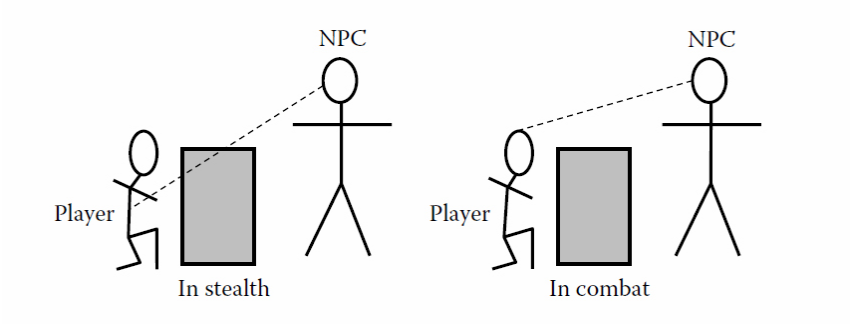
4.3 Kontrola viditelnosti

Kontrola viditelnosti se provádí pomocí paprsku a hledání jeho průsečíku s potenciální překážkou. Otázkou ale je, kam ten paprsek namířit. Zde je zase zapotřebí zamyslet se nad typem hry, kterou vyvíjíme. Pokud se jedná o akční hru, kde zrak slouží pouze k přibližné orientaci NPC, stačí použít jeden paprsek směrem ke středu hráče/postavy. Už tento jednoduchý systém poskytne NPC dostatek informací pro orientaci v souboji, zároveň ale umožňuje hráči obejít nepřítele a využít momentu překvapení. Pokud ale chceme sofistikovanější chování NPC nebo je naše hra více orientovaná na plíživý postup, je třeba vyslat více paprsků. Otázkou ale je, co bude cílová pozice těchto paprsků.

U hry The Last of Us, která je na plíživý postup velmi orientovaná, využili vývojáři poměrně jednoduchý, ale chytrý a hlavně fungující systém. To, kam se paprsek namíří, je dáno tím, jestli o hráči nepřátel ví nebo neví 6. Pokud je hráč skrytý nepřátelům a nebyl viděn, paprsek míří do středu hráče. Pokud ale hráče NPC uvidí a začne boj, paprsek míří na hráčovu hlavu. Díky tomuto systému je hráč během plížení zvýhodněn.

Vývojáři The Last of Us zkoušeli i komplexnější systém. Jejich původní verze vysílala paprsky téměř ke všem kostem hráče, přičemž každá kost měla nějakou váhu a výsledek se průměroval. Nicméně při testování zjistili, že je tento systém pro hráče matoucí a že hráči mají problém odhadnout, zda je NPC uvidí nebo ne. A tak i přesto, že to byl dobře navržený systém, z hratelnostního hlediska se vývojáři rozhodli pro jednodušší variantu.

To, že tento systém nefungoval v The Last of Us ovšem neznamená, že nemůže fungovat v jiné hře. Ve Splinter Cell: Blacklist byl využit hodně podobný systém. Hlavním mechanismem



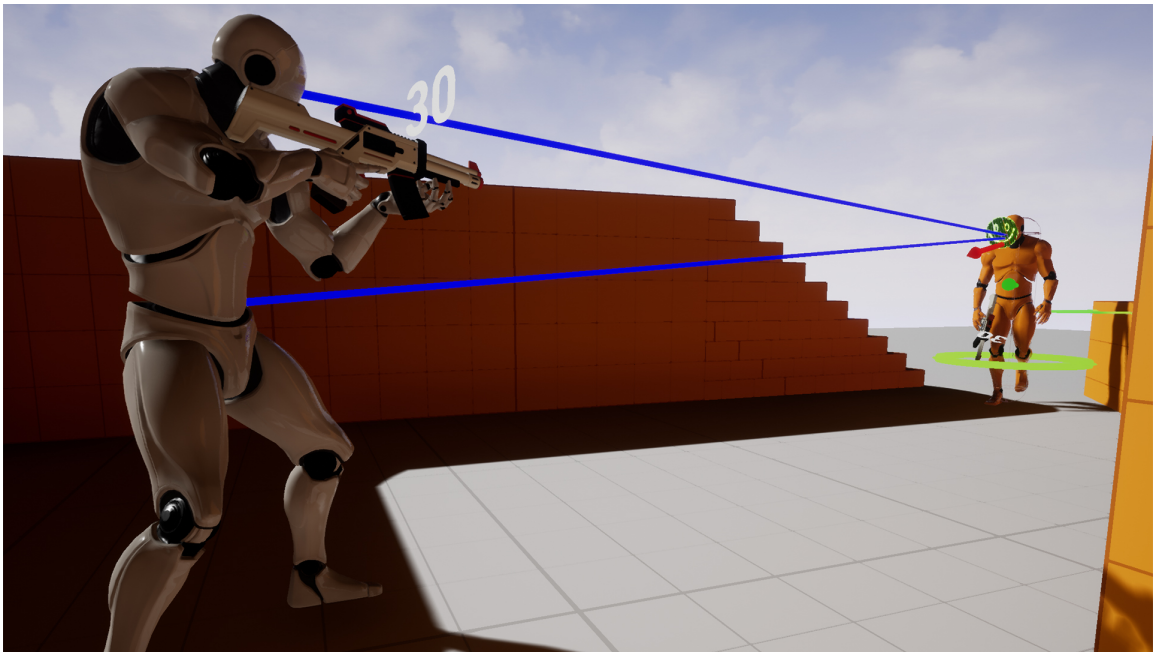
Obrázek 6: Viditelnost hráče [10]

této hry je právě plíživý postup úrovní a je na něj kladen daleko větší důraz než v The Last of Us. Cílová skupina této hry je tak na tento prvek více zvyklá a očekává větší výzvu. A tak vývojáři vytvořili systém, kde se paprsky vysílají k 8 bodům na hráči. Podle toho, v jaké póze se hráč nachází (stojící, přikrčený, ležící) se určilo, kolik kostí musí UI vidět, aby detekovalo hráče.

Při první implementaci zraku pro tuto práci, byl využit podobný přístup jako ve Splinter Cell: Blacklist. Vysílalo se několik paprsků od NPC ke hráči a podle toho, v jaké póze se hráč nacházel, se měnil minimální počet paprsků nutných k detekci hráče. Paprsky se vysílaly buď k bodům rovnoměrně rozděleným po kolizní kapsuli (v případě hraní z první osoby) nebo přímo k vybraným kostem hráčovy postavy (případě hraní z třetí osoby). Výsledek byl v tomto případě podobný, jako u The Last of Us. Často bylo obtížné rozpoznat, zda hráče NPC uvidí či ne. Navíc složitost tohoto systému komplikovala hledání chyb a optimalizaci. A proto byl tento systém nevhodný pro využití v této práci.

Kontrola viditelnosti byla proto zjednodušená. Výsledek je velmi podobný tomu, který byl nakonec použit i v The Last of Us. Bylo ovšem provedeno několik změn. Pokud hráč stojí, paprsky se vysílají do dvou míst: na hlavu a na trup 7. Stejně tak se vysílají dva paprsky i v případě, že hráč bojuje s NPC. Ukázkové úrovně jsou navrženy tak, aby byl hráč i ve skrčené poloze vždy o pár centimetrů vyšší než je nejnižší možný úkryt. NPC tak mohou hráče vidět i když je přikrčený za překážkou a hráč tak musí vynaložit značné úsilí, aby během boje dokázal uniknout nepřátelům. Pokud hráč nebyl zpozorován a bude přikrčený za překážkou, NPC si ho nevšimne. Vysílá totiž pouze jeden paprsek a to na hráčův trup. Jedinou možností jak může NPC odhalit hráče přikrčeného v úkrytu, je obejít hráče zezadu. Díky této implementaci kontroly viditelnosti je hráč při plíživém postupu zvýhodněn. A naopak v boji zase znevýhodněn.

Pro zlepšení hratelnosti, bylo při detekování hráče přidáno zpoždění. Pokud NPC hráče uvidí, objeví se u NPC indikátor, který se postupně naplňuje. Jakmile se naplní nad 70 procent, přepne se NPC do stavu Investigate. Jakmile se indikátor naplní na 100 procent, přepne se NPC do stavu Combat. Jednotlivé stavy ovlivňují barvu indikátoru. Ve stavu Investigate je indikátor žlutý a ve stavu Combat je indikátor červený. Toto zpoždění spolu s indikátorem u NPC umožňuje hráči poznat, zda jej NPC vidí a dává mu dostatek času na únik. Zároveň pokud hráč zaváhá, je toto zpoždění dostatečně krátké, aby hráče NPC zpozorovalo.



Obrázek 7: Vizualizace paprsků pro kontrolu viditelnosti

4.4 Optimalizace

Zrak implementovaný v této práci rozděluje vnímané objekty do tří kategorií: hráč, ostatní NPC a objekty. Pro každou kategorii objektů je vytvořen časovač, který opakovaně volá konkrétní funkci se zadanou prodlevou. Jelikož je každý typ detekovaného objektu řešený na vlastním časovači, můžeme individuálně měnit frekvenci. Například detekování hráče se může provádět 10 krát za sekundu, ale detekování objektů může být prováděno pouze 4 krát za sekundu. Tímto způsobem lze ušetřit výpočetní výkon. Navíc lze detekování pro každou kategorii vypnout. Můžeme tak vytvořit NPC, které bude detekovat pouze hráče. To nám dovoluje ještě více snížit náročnost tohoto systému. Vypínat detekování jednotlivých částí lze samozřejmě i za běhu hry.

Z organizačních a optimalizačních důvodů, je ukládání detekovaných objektů a postav rozděleno do několika částí: detekování hráče, přátelských NPC, nepřátelských NPC a objektů. Toto rozdělení nám dovoluje jednoduše přistupovat ke konkrétnímu typu objektů.

5 Sluch

Systém sluchu je stejně důležitý jako systém zraku. Oproti zraku je ovšem sluch používaný pouze jako orientační smysl. To znamená, že se sluch používá spíše k přibližnému určení nějakého jevu. Například strážný uslyší nějaký zvuk. Díky tomu se otočí přibližně směrem, odkud ten zvuk přišel, ale dále zpracovává informace zrakem. Jsou případy, kde může sluch přímo ovlivnit stav NPC. Například pokud bychom vytvářeli vojenskou základu a používali NPC jako hlídače, můžeme nastavit zvuk výstřelu jako automatický spouštěč poplachu, jelikož v tomto scénáři se taková reakce hodí.

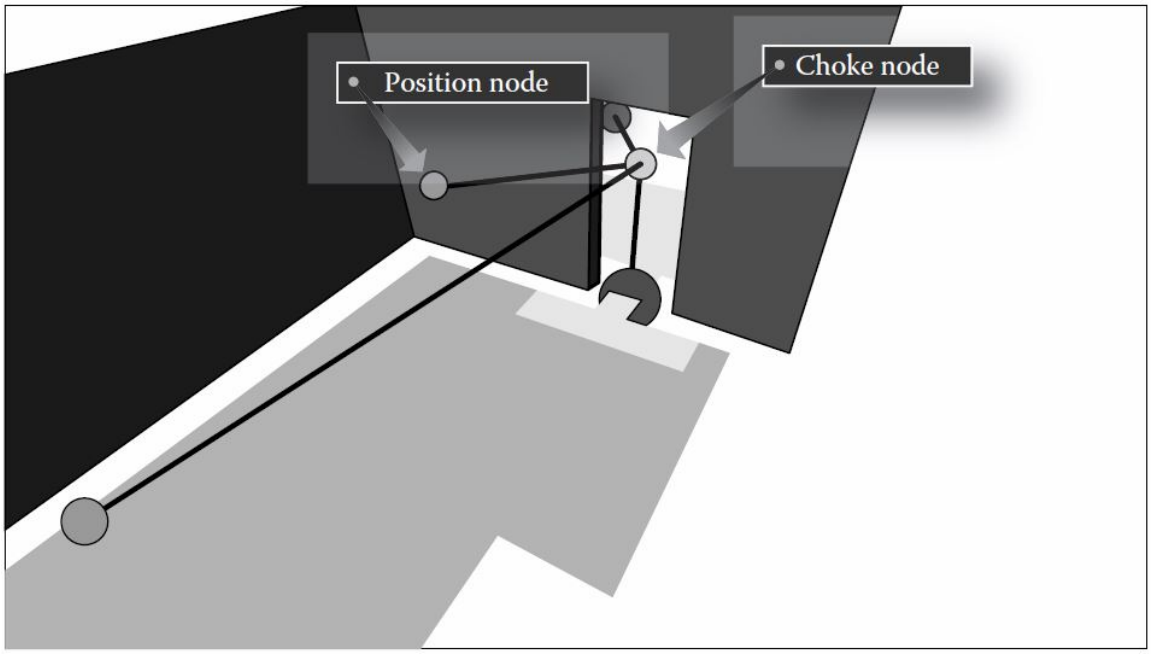
Implementovat zpracování zvuku se dá udělat mnoha způsoby a záleží na případu použití UI. Oproti zraku je sluch o něco problematičtější. Implementace zraku ve hrách je totiž víc přímočará. Provádí se tam sada poměrně jednoduchých kontrol, a pokud jsou splněný, NPC cíl vidí. U sluchu ovšem musíme počítat s šířením zvuku. Je jasné, že nebudeme implementovat fyzikálně realistické šíření zvuku, ale budeme se snažit přijít s nějakou aproximací nebo nějakým trikem.

Pokud bychom šíření zvuku neimplementovali a použili bychom například kontrolu vzdálenosti, NPC by vyhodnocovalo zvuk stejně bez ohledu na to, zda jsme v otevřeném prostoru, v interiéru nebo o patro výše. Výsledkem by bylo NPC, které slyší přes několik zdí, nebo přes několik pater stejně dobře, jako na otevřeném prostranství. Je tedy zapotřebí nějakým způsobem určit, jakým způsobem se zvuk šíří.

V reálném světě je šíření zvuku ovlivněno spoustou parametrů. Velikost interiéru, ve kterém se nacházíme, složitost (členitost) překážek, materiály, ze kterých jsou překážky vyrobeny, atd. Zvuk se odráží od překážek, přičemž část je absorbována a část putuje dále. Samozřejmě existují i prostory, které zvuk neutlumí, ale naopak zesílí. Je tedy jasné, že šíření zvuku je komplexní záležitost, a že reálie budeme muset při implementace velmi zjednodušit.

Při navrhování tohoto systému budeme vycházet z toho, jak se šíří zvuk v reálném světě. Pokusíme se najít ideální řešení pro implementaci a toto řešení postupně zjednodušovat až do fáze, kdy se dá použít v aplikaci běžící v reálném čase. Jednou z možností, jak takový systém navrhnout je použít systém velmi podobný vykreslovací metodě raytracing. Ze zdroje by se emitovaly paprsky všemi směry a při dopadu na plochu by se vypočítal směr odrazu a podle materiálu plochy i intenzita odraženého zvuku a celý proces by se opakoval. Tento proces by skončil po několika iteracích, nebo pokud by některý z paprsků dorazil k UI s požadovanou intenzitou. Bohužel je tento systém nepoužitelný ve hrách a interaktivních aplikacích. Vyžadovalo by to výpočet několika tisíců paprsků, což by mělo velký dopad na výslednou snímkovou frekvenci hry.

I přesto že je předchozí způsob implementace nevhodný, je dobrou nápovědou k lepšímu řešení. Lze totiž vidět, že potřebujeme pouze sadu paprsků, které procházejí různými otvory (dveře, okna, atd.). Takže pokud bychom dokázali identifikovat tyto místa, stačí nám pouze paprsky směřující do těchto míst.

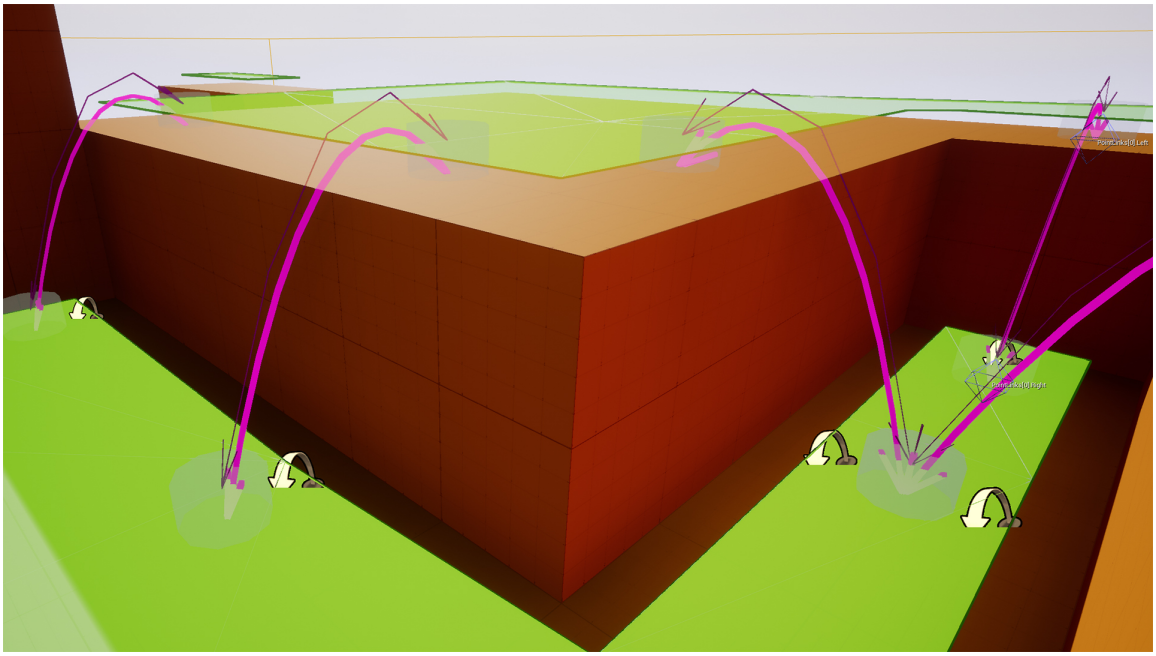


Obrázek 8: Využití choke nodes pro sluch ve Splinter Cell: Blacklist [10]

Ve hře Splinter Cell: Blacklist [12] použili vývojáři tzv. choke node (dále uzly). Tyto uzly byl umístěny do míst jako právě dveře nebo jiná průchozí místa8. Vývojáři vyvinuli tento systém pro jiný účel (pomocný systém navigace), ale dále ho využili právě pro výpočet šíření zvuku. Tyto uzly totiž propojují jednotlivá místa na mapě a lze je tak použít právě pro určení šíření zvuku. Jakmile se spustí nějaká zvuková událost, ze zdroje zvuku se najde nejbližší uzel a z toho uzlu se najde další nejbližší uzel, atd. Takovým způsobem určili vývojáři cestu zvuku prostorem a dokázali vypočítat výslednou intenzitu, a zda má NPC zvuk slyšet.

5.1 Implementace sluchu

V této práci se zaměříme na implementaci využívající navigační geometrii. Jedná se o systém, který využívá stejné navigační geometrie, jako NPC. Žádný z NPC se nedokáže pohybovat mimo navigační geometrii a proto můžeme využít této znalosti v náš prospěch. Místo vytváření uzlů mezi jednotlivými místnostmi/prostory, můžeme použít právě navigační geometrii pro nalezení cesty od zdroje zvuku k NPC. Tuto cestu je možno použít jako aproximaci cesty šíření zvuku prostorem. Dostaneme tak data potřebná k modifikaci intenzity zvuku. Nejdůležitější je samozřejmě vzdálenost. Pomocí utlumování intenzity zvuku se zvyšující se vzdáleností od zdroje, lze vytvořit nejzákladnější systém sluchu. Ovšem pokud chceme ještě přesnější reprezentaci šíření zvuku, můžeme výslednou intenzitu upravit i podle členitosti cesty zvuku. Jelikož nalezená cesta po navigační geometrii je obyčejné pole bodů, dokážeme tak určit úhly mezi jednotlivými úsečkami. Dá se tak simulovat útlum nebo zesílení zvuku pomocí odrazu.



Obrázek 9: Využití navlinků pro propojení vyvýšených mist.

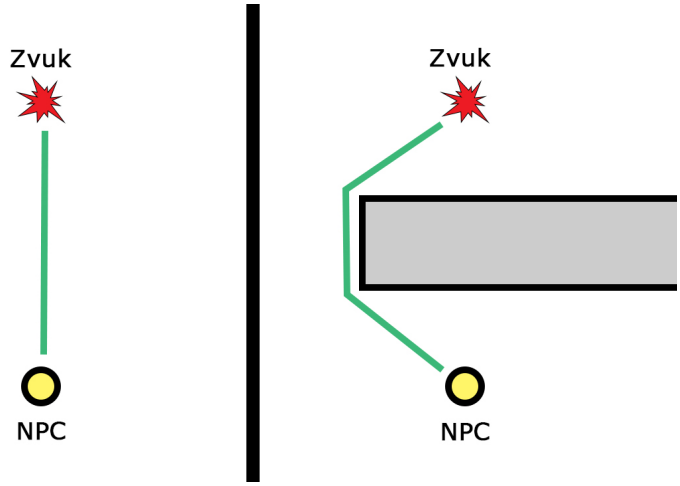
Tento způsob implementace šíření zvuku má ovšem jednu nevýhodu. Zvuk se může šířit pouze po navigační geometrii a tudíž je zde problém s šířením přes neprůchozí místa, nebo s šířením do výšky. Tento problém se dá vyřešit umístěním tzv. navigačních linků. Jedná se o objekt, který nám dovoluje spojit dvě částí navigační geometrie. Můžeme tak označit místo bez navigační geometrie jako průchozí. Pokud použijeme navigační linky v místech, jako jsou okna nebo pro vertikální spojení, můžeme tento nedostatek vyřešit 9. Je ovšem nutné pro tyto navigační linky vytvořit navigační filtr. Díky navigačnímu filtru můžeme tyto linky vyřadit z hledání cesty NPC.

Sluch v této práci se skládá ze dvou komponent. Noise komponenta je zodpovědná za vysílání zvuku. Hearing komponenta zvuk přijímá a zpracovává.

5.1.1 Noise komponenta

První, komponenta zvuku, je zodpovědná za vysílání zvuku. Jedná se o komponentu, které nastavujeme intenzitu a typ zvuku. Tato komponenta je přidaná na objekty, u nichž chceme, aby byly schopny vysílat zvuk, který dokáže NPC uslyšet. Je třeba zmínit, že zvuky slyšitelné NPC nemají nic společného s audio soubory. Nejedná se o zvuky, které slyšíme ve hře, nýbrž o signál, který říká NPC, že něco slyšel. Herní svět je totiž zaplněn různými zvuky, které jsou většinou pro správný chod NPC irelevantní.

Samotné vysílání zvuku je závislé na objektu. U zbraní je to při výstřelu, u hozených objektů je to při dopadu, u postavy je to při pohybu, atd. Tuto část je nutno naimplementovat individuálně pro objekty vysílající zvuk. Noise komponenta pak obsahuje funkci, která předá data o zvuku hearing komponentám konkrétních NPC.



Obrázek 10: Šíření zvuku.

Zvuk jsou rozděleny do tří kategorí:

- Normal – normální zvuk, který se používá pouze pro podpůrné systémy. Může to být zvuk nějaké neutrální činnosti, dialogu mezi NPC nebo ambientní zvuk. NPC si tohoto zvuku všimne, pokud zrovna nemá na práci nic jiného. Jedná se o typ zvuku, který oživuje herní prostředí a nutí NPC rozhlížet se po světě a reagovat kosmetickými akcemi.
- Suspicious – zvuk, který je pro NPC podezřelý a vyžaduje prozkoumání. Může se jednat o kroky hráče, které nepřítel uslyší nebo například dopad hozeného předmětu využitého pro rozptýlení nepřítele. Tento typ zvuku automaticky přepíná všechny nepřátelské NPC do stavu Investigate.
- Dangerous – zvuk, který je pro NPC nebezpečný. Jedná se primárně o zvuky jako střelba nebo například výbuch. V takovém případě každé NPC, které tento zvuk uslyší, automaticky přechází do stavu Combat.

5.1.2 Hearing komponenta

Hearing komponenta je zodpovědná za zpracování zvuku. Klíčovými faktory ovlivňujícími výsledné zvukové skóre jsou hlasitost zvuku a vzdálenost od zdroje zvuku. Zpracování zvuku tak probíhá dvěma způsoby podle toho, zda NPC zdroj zvuku vidí nebo ne. Pokud je zdroj zvuku viditelný, je vzdálenost vypočítána jako vzdálenost mezi zdrojem zvuku a NPC. Pokud ovšem NPC zdroj zvuku nevidí, musí aproximovat šíření zvuku v prostoru. Toho docílí nalezením cesty po navigační geometrii ze své lokace k zdroji zvuku. Vzdálenost této cesty pak určuje vzdálenost zvuku od NPC. Jedná se o poměrně jednoduchý, ale velmi efektivní způsob jak simulovat šíření zvuku prostorem. Zároveň zamezíme šíření zvuku přes zdi a jiné překážky 10.

U této komponenty je možné nastavit několik parametrů, které určí citlivost sluchu NPC. Hlavními parametry je Radius a Threshold. Radius určuje, na jakou vzdálenost NPC může

zvuk slyšet. Threshold určuje minimální skóre zvuku, které je slyšitelné pro UI agenta. Přepočet vzdálenosti a intenzity zvuku na výsledné skóre zvuku se pak dělá podle této rovnice 2.

$$n = \frac{distance}{radius} \tag{1}$$

$$s = (1 - n) \times loudness \tag{2}$$

Hearing komponenta má ale ještě další možnost jak upravit výsledné skóre zvuku. Využívá tzv. zvukové saturace. Jedná se o ovlivnění sluchu NPC okolními zvuky. Pokud bude NPC v tichém prostředí, všimne si i tichých zvuků. Pokud ale bude v hlučném prostředí, ostatní zvuky se v tomto hluku ztratí a NPC si jich nevšimne. Tento systém nám dovoluje v úrovních rozmisťovat tzv. saturační zóny, které nastavují parametr Saturation. Parametr Saturation pak snižuje výsledné skóre zvuku. Rovnice pro výpočet výsledné intenzity zvuku se pak změní takto3.

$$s = (1 - n) \times (loudness - saturation) \tag{3}$$

Jak si můžeme všimnout, předchozí rovnice jsou lineární. Existují případy, kdy to nemusí být ideální. Proto tato komponenta obsahuje možnost zpracování zvukového skóre exponenciálně. Pomocí následující rovnice 4 můžeme zvukové skóre modifikovat mírně při nízkých vzdálenostech, ale s přibývající vzdáleností se výsledné skóre začne snižovat intenzivněji. Tuto funkci můžeme použít například v otevřených prostorech, kde se vlivy ovlivňující zvuk násobí s přibývající vzdáleností.

$$y = \left(1 - \frac{(2^{an} - 1)}{(2^{a} - 1)}\right) \times (loudness - saturation), 0 \leq n \leq 1, a \neq 0 \tag{4}$$

Jakmile hearing komponenta zpracuje zvuk jako slyšitelný, zastaví se na určitou dobu, definovanou parametrem Cooldown, vnímání dalších zvuků. Jelikož je sluch primárně orientační smysl, který spíš pomáhá určit, kam se NPC má dívat, bylo by nevhodné, aby se při kontinuálním zvuku výsledek sluchu pořád aktualizoval. Pokud tedy NPC něco uslyší, sluch se na moment vypne. Mezitím NPC zareaguje na zvuk, který slyšel (například se podívá tím směrem) a až po uplynutí Cooldown doby se sluch znovu obnoví.

6 Rozhodování

Poku má UI aktuální informace o světě, musí se rozhodnout, jak na tyto informace reagovat. Rozhodovací část UI je zodpovědná za interpretaci informací a následné provedení akce. V UE pro tento účel můžeme použít jak BP skripty tak i C++, ovšem speciálním nástrojem vytvořeným právě pro navrhování logiky UI je behavior tree editor. A právě tento nástroj je použitý i v této práci.

6.1 Behavior tree

Hlavním nástrojem pro tvorbu logiky umělé inteligence v UE je behavior tree. Jedná se o stromovou strukturu složenou z několika druhů uzlů. Uspořádáním uzlů se pak určuje procházení tohoto stromu. Existuje několik teoretických modelů behavior tree a tak se nedá přímo určit standard. Ovšem jeden z nejpoužívanějších teoretických modelů BT využívá listové uzly jako akce a podmínky zároveň. Tok stromu je tak přímo určen úspěchem či neúspěchem prováděných akci a typem composite uzlu.

Unreal Engine používá odlišnou implementaci BT. Místo aby se strom prováděl s konstantní frekvencí, je využíván přístup založený na událostech. Strom se tak spouští pouze, pokud nastala nějaká změna vyžadující procházení stromu znovu. Navíc UE nevyužívá listové uzly jako podmínky. Místo toho jsou listové uzly (tzv. task uzly) využívány jako akce, kterou má UI provést a pro určení podmínek se využívají tzv. decorator uzly. Samozřejmě můžeme sestavovat BT i již zmíněným tradičním způsobem, kde tok určují task uzly, ovšem tím přicházíme o soustu výhod, které nám BT implementace v Unreal enginu nabízí.

6.1.1 Composite uzly

Coposite uzly určují, jakým způsobem se budou procházet podstromy. UE disponuje třemi typy composite uzlů:

- Sequence – provádí sekvenci uzlů zleva doprava. Pokud všechny uzly skončí úspěchem, skončí sequence uzel úspěchem. Jakmile jakýkoliv uzel neuspěje, celá sekvence neuspěje.
- Selector – provádí sekvenci uzlů zleva doprava. Při prvním úspěchu končí selector úspěchem. Pokud neuspějí všechny uzly, končí selector neúspěchem. Na rozdíl od Sequence uzlu, selector pokračuje po neúspěchu jednoho uzlu dalším uzlem v řadě.
- Simple parallel – Jedná se o uzel, který dokáže provádět dvě větve paralelně. Jedna z nich musí být obyčejný task uzel (s možnými decorator uzly) a druhá může být kompletní podstrom. Uzel může být nastaven tak, aby skončil, jakmile skončí hlavní task nebo aby po skončení hlavního Tasku počkal na dokončení druhé větve.

6.1.2 Task uzly

Task je uzel, který provádí konkrétní akce UI. UE disponuje několika základními task uzly jako například MoveTo(pohyb), Wait(čekání), atd. Můžeme si samozřejmě vytvářet i vlastní task uzly. Každý task musí mít určeno, zda je úspěšný či ne a proto je nutné po provedení akce i určit, zda byla akce úspěšná či ne. Pokud bychom tak neučinili, task by nikdy neskončil a průchod BT by byl narušen. Každý task může byt přerušen a tak je dobré vytvořit logiku i pro přerušení. Může se například jednat o navrácení modifikovaných hodnot do původní podoby.

Task uzly lze vytvářet pomocí c++ nebo Blueprintů. Zatímco implementace pomocí c++ může být výkonově šetrnější, implementace pomocí BP nám umožňuje rychlejší iteraci. Navíc s použitím funkcionality pro převedení BP do c++ se rozdíl výkonové náročnosti obou přístupů téměř smazává. Použití BP nebo c++ je proto čistě vývojářská volba.

6.1.3 Decorator uzly

Decorator je typ uzlu, který se používá ve spojení s task nebo composite uzlem. Jedná se o podmínkový uzel. Dovoluje nám tak usměrňovat průchod stromem. Navíc je možné u decoratoru nastavit chování přerušení. Pokud se podmínka změní, může decorator na změnu: nereagovat, přerušit sám sebe, přerušit uzly nižší priority, přerušit uzly nižší priority i sám sebe. Díky této funkcionalitě může BT ihned reagovat na změny dat. Na jeden task nebo composite uzel můžeme použít i více než jeden decorator. V takovém případě musí být podmínky všech decoratorů splněny aby mohl být proveden task nebo podstrom.

6.1.4 Service uzly

Service uzel se používá ve spojení s composite uzly. Jedná se o uzel, který periodicky provádí nějakou akci. Service uzel je aktivní, pokud je aktivní podstrom, ve kterém se nachází. Je možné určit frekvenci provádění service uzlu. Tento uzel se používá, pokud potřebujeme v nějakém intervale upravovat stav UI nebo zpracovávat data.

V ukázkové aplikaci byl service uzel použit například pro změnu rychlosti UI pomocníka při pohybu pomocí vodícího objektu. UI pomocník, který v tomto případě běží před hráčem, tak mění rychlost podle rychlosti hráče.

6.2 Rozhodování umělé inteligence nepřátel

Každé NPC má nějakým způsobem definovány stavy pro konkrétní situace. V případě jednoduché UI v akční hře to bude neutrální stav, kdy je hráč ještě daleko od NPC. Jakmile se hráč přiblíží, NPC se dostane do stavu boje a bude bojovat tak dlouho, dokud nepadne nebo neporazí hráče. I v tomto velmi zjednodušeném případě vidíme, že NPC má rozdělen svůj životní cyklus na dvě části. Každá tato část bude obsahovat vlastní pravidla pro chování NPC. V normálním

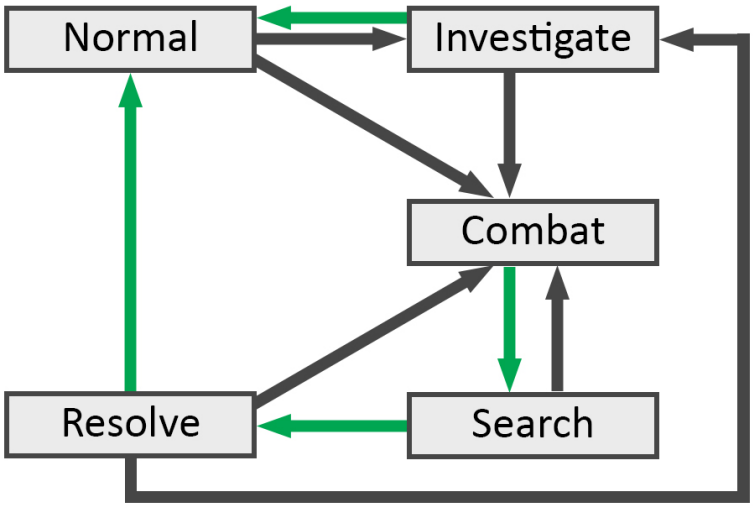
stavu může NPC hlídkovat, vést konverzaci s ostatními NPC nebo jen tak stát. V boji se ale NPC bude snažit krýt, obejít hráče nebo jej konfrontovat zblízka.

V této práci se zabýváme verzí nepřátel, kteří o hráči nevědí. Jejich UI je proto komplexnější. NPC na začátku své existence o hráči neví a provádí některou z neutrálních akcí. Během provádění těchto akcí, fungují veškeré smysly NPC popsané v předchozích kapitolách a tak může NPC uslyšet podezřelý či nebezpečný zvuk nebo na okamžik zahlédnout něco neobvyklého. V takovém případě se NPC rozhodne podezřelý jev prozkoumat. Pokud nic nenajde, vrátí se zpět do neutrálního stavu, ovšem může se stát, že při zkoumání narazí na něco nebezpečného, například na hráče. V takovém případě již NPC dál nezkoumá, ale začne s hráčem bojovat. Zde ovšem přichází změna oproti akční verzi UI. NPC totiž bude bojovat tak dlouho, dokud nepadne, nezdolá hráče nebo hráč neunikne. V posledním ze jmenovaných případů tak NPC bude chvíli hráče hledat a poté se vrátí zpět ke svým normálním povinnostem.

V ukázkové hře byl využit behavior tree, který je rozdělen na několik podstromů. Tyto podstromy odpovídají stavům nepřítele. V našem případě se jedná o pět stavů:

- Normal – je stav, kdy NPC provádí některou ze základních činností. Může se jednat například o hlídkování nebo postávání na místě s občasnými dialogy, atd.
- Investigate – do tohoto stavu se NPC dostane, pokud zaslechne podezřelý zvuk, nebo na okamžik zahlédne něco podezřelého. V takovém případě jde NPC prozkoumat lokaci, kde podezřelý jev zaznamenalo.
- Combat – je stav, kdy NPC bojuje s hráčem. To, jakým stylem bude bojovat, záleží na typu NPC. V ukázkové aplikaci jsou využity ofenzivní a defenzivní typy NPC. Ofenzivní typ se bude během útoku přibližovat ke hráči. Defenzivní typ NPC bude útočit z úkrytu.
- Search – do tohoto stavu se NPC dostane ze stavu Combat a to, pokud dostatečně dlouho neviděl nebo neslyšel hráče. NPC se pak snaží prozkoumat oblast, kde naposledy zaznamenal hráče. Oproti stavu Investigate stačí daleko menší impuls, aby se NPC vrátilo do stavu Combat.
- Resolve – je stav, který slouží spíše pro kosmetické účely. NPC během toho stavu mohou diskutovat o tom, co se právě stalo, nebo uklidit nepořádek, který v souboji vznikl. Tento stav slouží k více přirozenému přechodu z poměrně agresivního Search stavu do neutrálního Normal stavu.

Až na stav Normal, mají všechny stavy čas působení. Tyto časy lze nastavit individuálně pro konkrétní typ NPC. Navíc je možné nastavit i náhodnost časů pro jednotlivé stavy. Pokud tedy během aktuálního stavu nenastane impuls, který by časovač resetoval na původní hodnotu, stavy se postupným přepínáním vždy dostanou do počátečního stavu Normal. Automatické přepínání stavů je na na obrázku 11 označeno zelenými šipkami.



Obrázek 11: Diagram stavů UI nepřátel

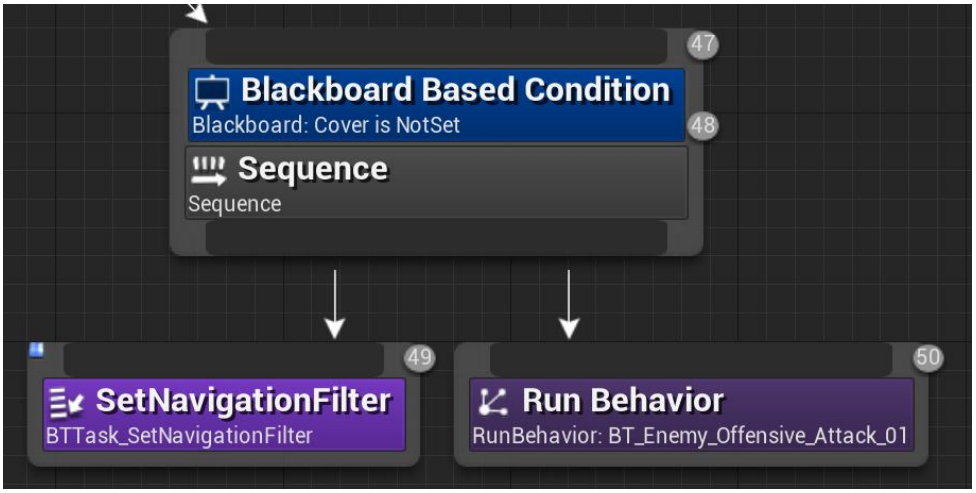
Stavy UI byly využity i při tvorbě behavior tree. Ten má jako kořenový uzel selector, jehož podstromy reprezentují jednotlivé stavy. Ty jsou seřazeny od nejvyšší priority pro tu nejnižší. Při resetování stromu se tak prochází podmínky jednotlivých podstromů a při prvním úspěchu se chod behavior tree zanoří do tohoto podstromu. Podstromy pro jednotlivé stavy mají decorator nekonečného cyklu. Tento decorator zajišťuje chod podstromu po dobu trvání daného stavu. Přerušení tohoto podstromu nastane změnou stavu UI. Jelikož jsou stavy řízené časovačem, vždy se postupně dostanou až na stav Normal.

Při tvorbě behavior tree byl využit modulární přístup, kdy se akce pro jednotlivé stavy vytvoří jako samostatné stromy, které jsou poté volány z hlavního stromu. Díky tomuto přístupu můžeme vytvářet různé kombinace behavior tree. Příkladem využití tohoto modulárního systému může být strom defensivního NPC pro stav Combat. Pokud toto NPC nenajde vhodný úkryt, přepne se do ofensivního módu, kdy se během útoku přibližuje ke hráči. Pro tento účel byl využit behavior tree, který byl původně vytvořen pro ofensivní UI 12.

V jedné z prvních verzí UI byl behavior tree použit i pro spouštění dialogů. Tento způsob spouštění dialogů se ale ukázal jako velmi limitující. Task uzel pro dialogy byl příliš složitý a obsahoval spoustu parametrů, které bylo vždy zapotřebí správně nastavit. Největším problémem ale bylo omezení spouštění dialogů u ostatních NPC, jelikož několik NPC mohlo používat stejný behavior tree a mohli se dostat do stejného stavu ve stejnou dobu.

6.3 Rozhodování umělé inteligence pomocníka

UI pomocník je typ UI, který má ve většině případů spíše kosmetickou úlohu. Může samozřejmě zasahovat do hry a aktivně hráčovi pomáhat, ovšem jeho hlavní úlohou je posílit iluzi virtuálního světa. Často se proto tento typ UI používá jako pomoc při vyprávění příběhu nebo



Obrázek 12: Vužití modulárního způsobu tvorby logiky UI.

pro navození konkrétní nálady. UI pomocníci mohou s hráčem vést dialogy během klidnějších pasáží hry. Při prozkoumávání virtuálního světa mohou komentovat hráčovy akce nebo mohou sami prozkoumávat svět a interagovat s objekty. V boji pak mohou hráči pomoct z téměř beznadějné situace a posílit tak atmosféru hry. Jsou to právě tyto drobné zásahy do hry, které vytvářejí z UI pomocníka postavu, na které vám bude záležet a tím se posílí i výsledný dojem ze hry.

A právě na tyto detaily se soustředí i implementace UI pomocníka v této práci. Souboj, prostorová orientace a spolupráce více agentů je demonstrována pomocí UI nepřátel. Pomocí UI pomocníka je v této práci demonstrováno více organické chování UI. Chování, které málokdy zasahuje do hratelnosti hry, ale o to víc se soustředí na posílení zážitku z virtuálního světa. Navíc oproti UI nepřátel, které se vyskytuje téměř exkluzivně ve hrách, lze UI pomocníka použít i v ostatních interaktivních aplikacích. Lze jej použít například jako virtuálního průvodce.

I přesto, že řada postupů použitých u UI nepřátel je shodná s těmi použitými u UI pomocníka, výsledná logika se velmi liší. Behavior tree UI pomocníka ani zdaleka tolik nespoléhá na stavy UI a je orientován spíše kolem akcí hráče. Daleko více podmínek se soustředí na pozici nebo rychlost pohybu hráče, aby dokázal adekvátně na tyto změny reagovat. Nechceme totiž hráče zdržovat. Pokud chce hráč prozkoumávat herní svět, UI pomocník bude na toto chování adekvátně reagovat a bude prozkoumávat s hráčem. Pokud chce ale hráč rychle herní úrovní proběhnout, nechceme aby ho UI pomocník zdržoval.

6.3.1 Implementace bodu zájmů a smart zóny

Body zájmu jsou speciální objekty, kterými můžeme určit místa zajímavá pro UI pomocníka. Pokud je UI pomocník ve stavu, kdy se nenachází ve vypjaté situaci nebo nemá nic na práci, může tyto body vyhledat a interagovat s nimi. Tyto body obsahují informace o typu interakce, které s nimi může UI pomocník mít. Mezi tyto interakce patří například dialog, animace nebo

prostě jen to, že si UI bude konkrétní objekt prohlížet. Zároveň můžeme určit, zda se jedná o jednorázovou interakci a nebo zda se může interakce opakovat. Navíc můžeme nastavit i časovač, který zamezí jakékoliv interakci s tímto objektem, dokud nedoběhne dokonce.

Smart zóna je typ objektu používaný pro spouštění akci UI pomocníka. Jedná se o zónu, která udržuje reference všech bodu zájmů, které se nacházejí uvnitř této zóny. Tímto způsobem můžeme optimalizovat vyhledávání potenciální bodu zájmů. Smart zóna má ovšem i další výhody. Jednou z hlavních výhod je možnost detekovat přítomnost UI pomocníka. Pokud je UI pomocník v neutrálním stavu a bude procházet smart zónou, automaticky se začne vyhodnocovat, zda některý z bodu zájmů uvnitř zóny UI pomocníka nezaujme. Může tak například nastat situace, kdy hráč a UI pomocník procházejí úrovní a ocitnou se ve smart zóně, načež UI pomocník uvidí něco zajímavého, upozorní na to hráče a vydá se objekt prozkoumat.

Funkcionalita smart zóny se ovšem neomezuje pouze na práci s body zájmu. Smart zóna může fungovat i jako jednorázový spouštěč dialogů. Dokáže spouště dialogy manuálně zadané, ale i dialogy z XML souboru UI pomocníka. Funkcionalita smart zón lze samozřejmě rozšířit pomocí BP skriptů.

7 Pohyb UI

Pohyb je nedílnou součástí humanoidní UI. Aby mohl NPC reagovat na potenciální hrozbu, musí umět doběhnout do úkrytu. Stejně tak pokud potřebuje NPC prozkoumat okolí, musí umět se pohybovat v herním světě. Pro implementaci humanoidní UI je tedy zapotřebí nějaký systém, který nám dovoluje ovládat pohyb NPC.

7.1 Navigační systém

Navigační systém umožňuje plánování a provedení pohybu NPC po úrovni. Určením lokace můžeme získat cestu, kterou následně NPC půjde. Můžeme ovšem získat i informace, pokud je cíl nedosažitelný. Navigační systém nám také umožňuje upravit cenu konkrétních lokací, což má za následek nalezení jiné, optimálnější cesty.

7.1.1 Navigační geometrie

Unreal engine používá pro hledání cest navigační geometrii. Jedná se o geometrii pokrývající veškeré plochy, po kterých může NPC chodit, nacházející se v Navigation volume. Pro generování této geometrie používá UE open source knihovnu Recast [13]. Tato metoda převede geometrii herní úrovně do voxelové reprezentace. Tímto způsobem lze detekovat překážky a najít lokace, po kterých může NPC chodit. Tyto lokace jsou následně rozděleny na regiony a poté jsou aplikovány filtry, které upraví tyto regiony podle parametrů definujících NPC, jako například šířka NPC. U takto upravených regionů se vygenerují kontury, čímž se členitost regionů velmi zjednoduší. Poté jsou tyto regiony triangulovány. Výsledkem je navigační geometrie, která nám určuje veškeré lokace, po kterých může NPC chodit.

7.1.2 Nav area

Nav area je objekt, který nám dovoluje pozměnit ohodnocení navigační geometrie. NPC, při vyhledávání cesty po navigační geometrii, hledá cestu s nejmenší cenou. Za normálních okolností, kdy je použit pouze navigační geometrie, se jedná o nejkratší cestu. Ovšem ne vždy budeme chtít, aby UI chodila co nejkratší cestou. Existují případy, kdy potřebujeme určit jinou cestu, jako tu primární a nemusí se jednat o nejkratší cestu. V takovém případě můžeme využít nav area třídu, díky které můžeme manuálně upravit ohodnocení určité části navigační geometrie. Takto vytvořenou nav area pak použijeme na nav mesh modifier. To je objekt, který umístíme do scény a který reprezentuje fyzické rozměry námi používané nav area.

7.1.3 Nav filter

Nav filter je třída, která nám dovoluje vytvořit sadu pravidel ovlivňujících navigaci NPC ve světě. Tato pravidla jsou určená pomocí nav area (popsané v předchozím odstavci). Nav filter může obsahovat jednu a více nav area, díky kterým můžeme manuálně upravit hledání cesty

po navigační geometrii. Oproti nav area se nav filter používá například u kontroléru postavy, případně pak můžeme specifikovat nav filter i pro jednotlivé akce. Typickým příkladem může být MoveTo task v behavior tree. Můžeme použít konkrétní filtry pro jednotlivé MoveTo uzly aniž bychom tyto filtry museli aplikovat přímo na kontrolér.

V této práci je použit nav filter pro několik účelů. Jedním z nich je vyfiltrování pomocných nav linků použitých pro šíření zvuku. Tyto nav linky jsou pomocí filtrů vyřazeny z hledání cest pro NPC. Dalším případem je použití speciálního filtru, při boji s nepřáteli. Tento filter má navíc přidanou jednu Nav Area. Ta je využita na hráči a reprezentuje oblast před hráčem. Tato oblast je během boje vyfiltrovaná a UI tak nebude nesmyslně běhat přímo před hráčem. Místo toho se ho bude vždy snažit obejít, případně najde lepší cestu do úkrytu.

7.1.4 Nav link

Nav link nám umožňuje spojovat části navigační geometrie v místech, kde navigační geometrie chybí a kde je místo pro NPC neprůchozí. Navigační linky se používají i ve spojení s dalšími objekty. Pokud budeme chtít vytvořit systém pro přeskakování překážek, použijeme k tomu kromě spouštěcích objektů i nav link, aby NPC vědělo, že je toto místo průchozí. V této práci jsou nav linky použity i při implementaci sluchu. Slouží k propojení vyvýšených míst nebo k přemostění překážek, aby se mohlo správně simulovat šíření zvuku.

7.2 Pohyb umělé inteligence nepřátel

U nepřátel je pohyb velmi důležitý. Nepřátele totiž s hráčem většinou nekomunikují. Hráč s nimi nevede konverzaci a kromě boje s nimi ani nijak jinak neinteraguje. Proto je pohyb jedním z hlavních způsobů, jak nepřítel dokáže komunikovat svůj záměr. NPC by tedy mělo při útoku volit správnou cestu, ze které dokáže hráč poznat, že na něj NPC útočí. Při obraně by mělo NPC nejen najít správný úkryt, ale i zvolit správnou cestu do úkrytu. Pokud NPC najde výborný úkryt, ale při přesunu proběhne přímo před hráčem, nejen že bude vypadat hloupě, ale zároveň vytrhne hráče ze zážitku.

7.2.1 Omezení pohybu nepřátel

Jeden z prvních problémů, které je třeba vyřešit, je omezení pohybu NPC. Nejzákladnějším způsobem je samotná geometrie úrovně. V některých případech je možné upravit rozložení objektů v úrovni tak, abychom vytvořili překážku, které usměrní NPC požadovaným směrem. Toto je ovšem spíše vzácný případ. Při designování úrovní se totiž v první řadě myslí na hratelnost a tak není možné měnit rozmístění prvků v úrovni dle libosti. Je tedy třeba využít jiný systém. Nejčastěji se jedná o systém, který mění cenu konkrétních lokací na navigační geometrii. NPC totiž vyhledává nejlevnější(nejkratší) cestu po navigační geometrii a pokud dokážeme nějakým způsobem ohodnotit určité lokace vyšší cenou, NPC se bude snažit této lokaci vyhnout.

V ukázkové aplikace byla pro tento účel využita nav area. Nav area byla využita například při označení zóny, kde se může ukrýt hráč. Bylo by totiž nežádoucí aby touto zónou NPC procházelo. Dalším využitím bylo omezení pohybu některých nepřátel. Přiřazením specifické nav area můžeme omezit pohyb NPC pouze na tuto oblast. Tato funkcionalita se hodí, pokud chceme, aby NPC zůstalo na svém stanovišti. Může se jednat třeba o strážnou věž. Speciálním případem využití nav area je aproximace hráčova zorného pole. Jedná se o obdélníkovou zónu připojenou na hráče. Tato zóna má přiřazenou NavArea se zvýšenou cenou, takže ohodnocuje oblast před hráčem. NPC tuto zónu ignorují, pokud s hráčem nebojují. Ovšem jakmile se přepnou do stavu boje, využívají ohodnocení této zóny pro úpravu cesty. Díky tomuto systému můžeme omezit případy, kdy NPC proběhne přímo před hráčem.

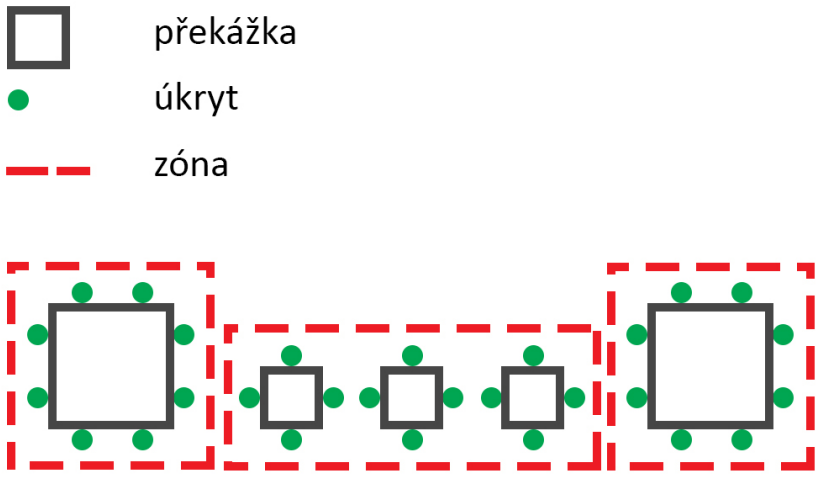
7.2.2 Pohyb více nepřátel

Málokdy se bude v úrovni objevovat pouze jeden nepřítel. Proto je zapotřebí vytvořit i logiku pro koordinovaný pohyb. Ať už se jedná a vojenskou jednotku ve válečné hře, civilisty ve městech nebo skupinu NPC spolupracující s hráčem, v každém z těchto případů musí NPC brát ohled i na ostatní NPC. Způsobů koordinace pohybu mezi NPC je spoustu. Od nejjednodušších, kdy si NPC pouze předávají informace, které upravují průchod rozhodovacím stromem až po složité taktické manévry vojenských jednotek. Stejně jako v předchozích případech je tedy zapotřebí zamyslet se nad použitím NPC.

Poměrně často se ve hrách setkáme s věšti skupinkou nepřátel. Pokud hráč v takovém případě spustí poplach, musí se nepřátele nějakým smysluplným způsobem rozmístit. Bylo by nežádoucí, pokud by nepřátele našli pouze nejbližší úkryt a všichni se nahromadili za jednou překážkou. Samozřejmě existuje spoustu způsobů jak tento problém vyřešit. NPC mohou například hledat úkryty jeden za druhým a jakmile jeden NPC najde úkryt, označí jej za použitý a veškeré úkryty v určité vzdálenosti již nebudou moct být použity. Dalším řešením může být i rozmístění NPC kolem hrozby a teprve pak spustit hledání úkrytů. Způsobů, jak tento problém vyřešit je spoustu a řešení se budou lišit podle počtu nepřátel a velikosti úrovní.

V ukázkové aplikaci byly pro efektivní hledání úkrytu vytvořeny speciální zóny 13. Tyto zóny jsou do úrovně umístěny a vytvarovány designerem. Jejich účelem je rozdělení úrovně do logických celků. Například pokud bychom měli úroveň s budovou, zóny by mohly reprezentovat jednotlivé místnosti. Každá zóna si uloží reference na úkryty nacházející se uvnitř. Pokud NPC hledá úkryt, najde si nejdříve vhodnou zónu a teprve poté vyhledává vhodný úkryt. K tomu využije právě pole referencí na úkryty nacházející se uvnitř konkrétní zóny. Pokud ani jeden z těchto úkrytů nesplňuje požadavky, NPC si zvolí druhou nejlepší zónu a proces hledání úkrytu opakuje.

Tyto zóny mají několik parametrů, podle kterých je NPC může filtrovat. Jedním z hlavních parametrů je maximální povolený počet NPC v dané zóně. Každá zóna si udržuje informaci o aktuálním počtu NPC nacházejících se uvnitř. Pokud je tento počet stejný jako maximální povolený počet NPC, jakékoliv další NPC tuto zónu při hledání úkrytu přeskočí. Tímto způsobem



Obrázek 13: Vyhledávání úkrytu pomocí zón

se NPC automaticky a velmi přirozeně rozdělí a najdou si úkryty na větší ploše. Nestane se tak, že by se větší počet nepřátel nahromadil za jednou překážkou.

7.3 Pohyb umělé inteligence pomocníka

Pohyb UI pomocníka je oproti nepřátelům o něco složitější. UI pomocník je totiž daleko častěji poblíž hráče a ten si velmi rychle všimne jakýchkoliv chyb a anomálií pohybu UI. Navíc od UI pomocníka očekáváme větší inteligenci a tudíž i chytřejší pohyb v prostoru. V mnoha hrách se UI pomocník používá i jako průvodce a tak je třeba vyřešit nejen následování hráče, ale i pohyb před hráčem.

7.3.1 Pohyb po křivce

Pohyb po křivce je nejjednodušší varianta jak zajistit pohyb před hráčem. UI využívá pro pohyb předem připravenou křivku a pouze hlídá, zda je před hráčem. Tento způsob pohybu UI se používá od prvních her s UI pomocníky. Jeho hlavní výhodou je fakt, že víme přesně, po jaké trase NPC půjde. Navíc je NPC napevno svázané s touto křivkou a jakékoliv fyzikální objekty pro něj nejsou překážkou, takže se nemůže zaseknout. Bohužel má tento způsob v dnešní době spíše více nevýhod. NPC má pouze jednu možnou trasu a proto je tento způsob vhodný primárně pro velmi lineární úrovně. Často tento pohyb vypadá velmi nepřirozeně. A jeden z nejhorších problému je, když se hráč postaví na místo, kde se má zastavit NPC. Často to končí zaseknutím hráče a nutností restartovat úroveň. I přes všechny tyto nevýhody se pořád najdou případy, kdy je tento systém tím nejvhodnějším.

7.3.2 Pohyb po bodech

Podobně jako v případě pohybu po křivce je pohyb po bodech předem daný. Body jsou umístěny designerem a cesta NPC je tak předem daná. Výhodou oproti pohybu po křivce je to, že samotný pohyb už není určený napevno. NPC se pohybuje pomocí navigačního systému od bodu k bodu. Díky tomu můžeme mít v úrovni dynamické objekty, jelikož se jim NPC dokáže vyhnout. Navíc se tento systém dá daleko lépe přizpůsobit pro více otevřené úrovně.

7.3.3 Pohyb s vodícím objektem

Jeden z nejjednodušších způsobů, jak zajistit, aby se NPC pohybovalo před hráčem je vytvořit vodící objekt. Jedná se o pomocný objekt, který se pohybuje spolu s hráčem. Tento objekt je umístěn před hráčem ve směru jeho pohybu. NPC pak používá vodící objekt jako svůj cíl, ale jelikož se vodící objekt pohybuje spolu s hráčem, NPC ho nikdy nedohoní. Poměrně důležitým vylepšením je zavedení interpolace pohybu vodícího objektu. Vytvoří to plynulý pohyb s mírným zpožděním, takže při zatáčení hráče se vodící objekt nepřesune najednou o velkou vzdálenost (což by způsobilo škubnutí NPC), ale plynule se přesune na požadované místo. Zároveň se doporučuje měnit vzdálenost vodícího objektu od hráče podle toho, jak rychle hráč jde. Tento systém vyžaduje i úpravu samotného NPC. Hráč se totiž může otočit o 180 stupňů a NPC se ocitne za hráčem. Proto je zapotřebí nastavit NPC tak, aby v případě, že se nachází za hráčem, běželo daleko rychleji a jakmile se dostane na hráčovu úroveň, zpomalilo na rychlost dostatečnou k předběhnutí hráče. Posledním vylepšením je ošetření případů, kdy je vodící objekt nedosažitelný, případně zajistit, aby byl vždy dosažitelný. Nejjednodušším řešením tohoto problému je aktuální pozici vodícího objektu promítat na navigační geometrii.

7.3.4 Kombinace pohybu po bodech a vodicího objektu

Kombinací dvou předešlých systému dokážeme pohyb NPC ještě vylepšit. Samotné následování vodícího objektu funguje dobře, ovšem mohou nastat případy, kdy je hráč nucen často měnit radikálně směr (např. interiéry) a pohyb NPC v takovém případě nemusí vypadat velmi přirozeně. Dodatečná informace v podobě navigačních bodů nám ale dovolí lépe analyzovat situaci a přizpůsobit tak pohyb NPC. To, že máme předem určené cesty pomocí navigačních bodů, neznamená, že se NPC musí pohybovat pouze po této cestě. Pořád můžeme využívat vodícího objektu jako primární systém pro navigaci, ale změna pozice vodícího objektu může využívat dodatečné informace poskytnuté navigačními body.

V této práci je tento systém ještě vylepšen o dynamicky generovanou cestu. Místo abychom pro posun vodícího bodu používali pevně danou cestu určenou předem umístěnými body, využíváme navigačního systému a cestu od hráče k cíli generujeme. Tímto způsobem můžeme generovat cestu k cíli z jakéhokoliv místa v herní úrovni. Jakmile se tedy vydáme směrem k cíli, vygeneruje se cesta k cíli a vodící objekt se bude pohybovat po této cestě. Pohyb NPC před hráčem je tak možný kdekoliv v úrovni.

Tato varianta je samozřejmě výkonově náročnější, než varianta s pevně zadanými cestami. Je tedy nutné celý proces posunu vodícího objektu optimalizovat. Ten se skládá ze tří částí. Nejdříve musíme vygenerovat cestu od hráče k cíli, poté najít nejbližší bod na této cestě a nakonec provést posun vodícího objektu. Vygenerování cesty je nejnáročnější část tohoto procesu. Proto generujeme cestu jen na začátku hry nebo pokud se hráč od cesty dostatečně vzdálí. Jakmile je cesta vygenerovaná, musíme najít nejbližší bod ke hráči. Procházíme tedy jednotlivé úsečky cesty a hledáme nejbližší bod 5 14. Abychom optimalizovali posun vodícího objektu po cestě, uložíme si při vyhledávání nejbližšího bodu i index úsečky, na které se nachází. To nám umožní při výpočtu posunu vodícího objektu vynechat kus cesty, který předcházel této úsečce, jelikož vodící bod posouváme dopředu.

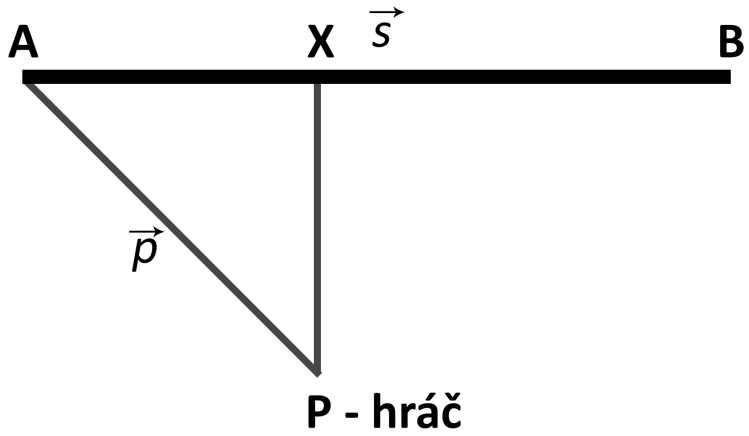
$$\overrightarrow{s} = B - A \tag{5a}$$

$$\overrightarrow{p} = P - A \tag{5b}$$

$$d^2 = ||\overrightarrow{s}||^2 \tag{5c}$$

$$t = \frac{\overrightarrow{s} \cdot \overrightarrow{p}}{d^2} \tag{5d}$$

$$X = \overrightarrow{a} \times t + A \tag{5e}$$



Obrázek 14: Nalezení nejbližšího bodu na cestě.

Aktuální úsečka cesty zároveň slouží i pro testování směru pohybu hráče. Pro správnou reakci UI pomocníka totiž potřebujeme znát i to, zda hráč jde po cestě směrem k cíli. K tomuto účelu nám nestačí získat jen směr k cíli, jelikož ten nereprezentuje směr cesty.

8 Animace

Animace jsou nejpoužívanějším způsobem jak prezentovat rozhodování NPC. U humanoidních NPC je tento způsob prezentace chování navíc posílen tím, že odráží reality světa, ve kterém žijeme. Všichni jsme seznámení s řečí těla a různými grimasami. I ty nejjemnější pohyby dokáži komunikovat spoustu informací a to právě proto, že se s nimi setkáváme denně. Kvalita animací je každým rokem na vyšší úrovni a tak si mohou vývojáři dovolit využívat stále jemnější detaily.

8.1 Animation blueprint

V Unreal enginu se pro práci s animacemi používá animation blueprint (dále ABP). Je to typ blueprintu, který se specializuje na zpracování animací. Máme zde možnost pracovat s kostrou postavy, jednotlivými animacemi a hlavně vytvářet logiku pro spouštění, prolínání a přepínání animací.

8.1.1 Event graph

Event graph je zodpovědný za příjem a zpracování dat potřebných k správnému chodu ABP. Zde získáváme například informace o rychlosti postavy, směru natočení nebo zda je postava přikrčená. Všechny toto data si můžeme uložit a použít je v anim graphu, state machine, blendspace, atd. Event graph podobně jako v případě klasických BP obsahuje několik základních eventů. Animation construct je spouštěn na začátku ABP. Animation update (který budeme používat nejvíce) funguje stejně jako tick v klasických BP, tzn., že se spouští každý snímek.

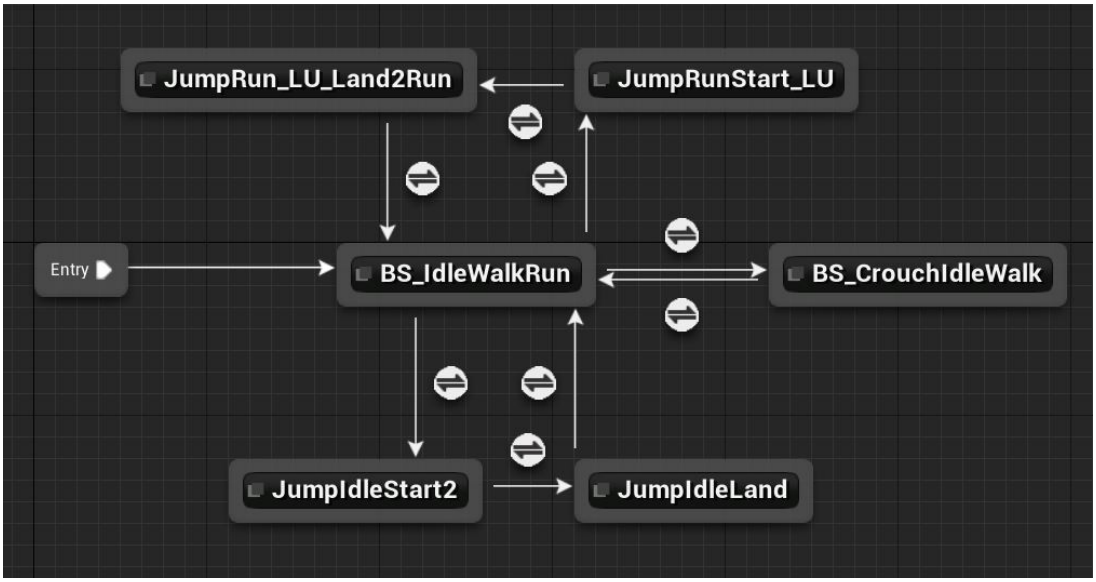
8.1.2 Anim graph

Anim graph nám umožňuje prolínat, přepínat a kombinovat animace. Ke správnému chodu je zapotřebí nejdříve získat informace, na základě kterých budeme provádět operace nad animacemi. Tyto informace získáme v event graphu.

V této práci byl anim graph využit například u animace nepřátel. Ti využívají jinou sadu animací pro klidový stav a jinou sadu animací pro boj. V každém z těchto stavů musí být nepřítel schopen pohybovat se různými rychlostmi a tudíž i přehrávat různé animace. Do těchto animací je navíc vložena možnost rozhlížet se a mířit. NPC tak může i za běhu sledovat hráče a případně na něj i mířit. Navíc bylo nutné upravit graf tak, aby bylo možné spouštět animace i z jiných částí logiky UI. Typickým příkladem jsou animace střelby z úkrytu, které jsou spouštěny z behavior tree.

8.1.3 State Machine

State machine 15 se v ABP používá pro prolínání větších celků animací. Můžeme v něm prolínat jak jednotlivé animace, tak i blendspace. Dovoluje nám vytvářet vazby mezi konkrétními animacemi a určovat podmínky, za jakých se animace přepne na konkrétní uzel. Díky vazbám tak

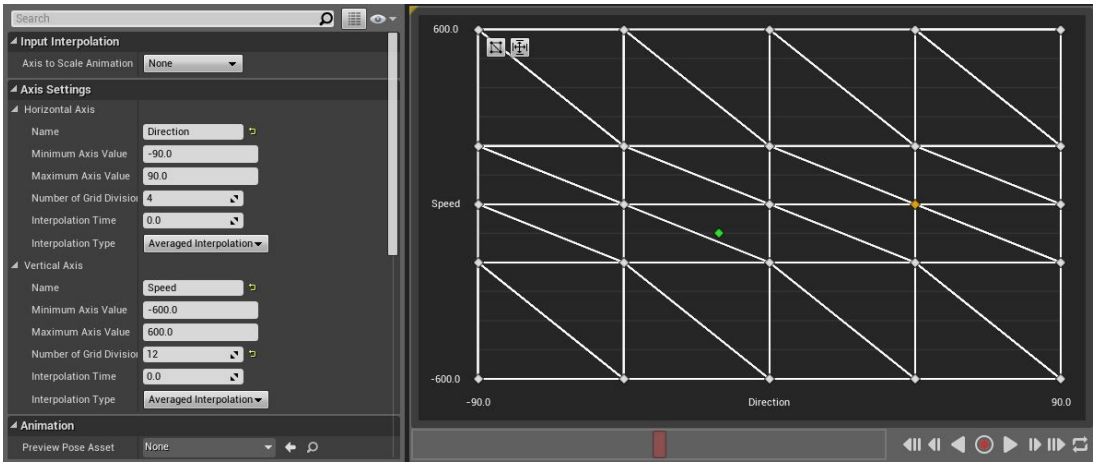


Obrázek 15: Ukázka jednoduché state machine

nemůžeme přecházet z konkrétního uzlu pouze na námi definované uzly a za námi definovaných podmínek. Dá se tak vytvořit sekvence animací, která se má provést za určitých podmínek a v určitém pořadí. Jako příklad uvedeme skok postavy. Ten se skládá ze tří částí: výskok, let a dopad. Výskok tedy provedeme, pokud je stisknuto tlačítko pro skok. Z něj se pak po jeho dokončení přesuneme na animaci letu a v této animaci setrváme, dokud je postava ve vzduchu. Jakmile postava přestane být ve vzduchu (kolizní objekt postavy je zpět na zemi), přehrajeme animaci dopadu. Díky tomuto systému je tak možné provádět plynulou animaci skoku z různě vysokých překážek. Samozřejmě můžeme vytvořit komplexnější systém, kdy budeme rozlišovat mezi skokem na místě, za chůze nebo za běhu.

8.2 Blendspace

Blendspace je typ objektu v UE, který nám dovoluje plynule prolínat animace. Existuje ve dvou verzích: 1D a 2D. Jak již názvu napovídají, první z nich je pouze jednorozměrný, zatímco ten druhý je dvourozměrný. V editoru je blendspace reprezentován plochou (úsečkou v případě 1D blendspacu), na kterou lze umísťovat animace 16. Této ploše určíme rozsah a názvy os. Samotný blendspace pak využívá dat, získaných v event grafu, aby plynule prolínal mezi jednotlivými animacemi. Jedná se o velmi efektivní způsob jak vytvořit plynulé přechody mezi sadami animací. Například u chůze můžeme mít osm různých animací (pro osm směrů) a proto můžeme vytvořit blendspace, který bude mít na jedné ose rychlost a na druhé směr pohybu. Podle těchto dat pak můžeme získat animaci, která plynule prolíná mezi námi zadanými osmi animacemi.



Obrázek 16: Blendspace - nastavení a editor

8.3 Aim offset

Aim offset je speciální typ blendspace. Na první pohled vypadá identicky. Také má 1D a 2D verzi. Aim offset má ovšem jiné využití. Používá se pro prolínání póz, které jsou následně vrstveny na další animace. Typickým příkladem může být otáčení hlavou. V mnoha případech chceme mít možnost otáčet hlavou a sledovat nějaký cíl. Zároveň ale chceme, aby zbytek těla prováděl ostatní animace, jako například chůzi. A právě k tomu se využívá Aim offset. Je tvořený jedno snímkovými animacemi (pózami) reprezentujícími extrémy. Tyto pózy jsou poté nastaveny additive blending, což nám umožňuje je vrstvit na ostatní animace.

Aim offset se používá ve spoustě případů. Například v této práci byl aim offset použit pro rozhlížení NPC, ale také pro míření. V event graphu je nejdříve spočítán úhel, kterým by mělo NPC mířit a ten je poté použit jako vstup pro aim offset. Je to jeden z případů, kdy animace není pouze vizuální částí systému, ale také aktivně ovlivňuje funkcionalitu systému.

8.4 Typy animací

Ve hrách se používají primárně animace, kde se postava pohybuje na místě. Například animace chůze vypadá, jako by postava klouzala na místě. Animace jsou tímto způsobem vytvořeny, jelikož samotný pohyb zprostředkovává postava a animace funguje pouze jako vizuální aspekt postavy. Tyto animace jsou pak ovládány pomocí různých systémů, které využívají data postavy jako např. rychlost nebo směr pohybu. V případě Unreal enginu se jedná o ABP.

Existuje ovšem i typ animací nazvaný root motion. Tyto animace mají v kořenové kosti uložený pohyb animace (od toho název root motion), který je využit pro posun postavy. Tento typ animací je velmi výhodný například při interakci s prostředím, objekty nebo postavami. Dokážeme totiž přesně určit pozici, kde se má animace přehrát. Dá se tak velmi jednoduše vytvořit například animace lezení nebo interakce mezi dvěma postavami.

V ukázkové animaci byly root motion animace využity například při střelbě nepřátel z úkrytu. Úkryty jsou umístěny tak, aby byla vzdálenost ke hraně překážka vždy přibližně stejná, jako vzdálenost pohybu animace. Samotná animace pak poskytne data pro posun postavy.

8.5 Montage

Montage je další způsob jak skládat animace dohromady, ovšem proti State machine se jedná o permanentní spojení animací. Montage je tvořen jednou nebo více animacemi, které jsou přehrány v námi určeném pořadí. Můžeme je používat v místech, kde chceme přehrát více animací najednou jako celek. Příkladem může být šplháni na překážku. To se může skládat z výskoku, zachycení se o hranu překážky a vyhoupnutí se nahoru. My chceme aby se tyto animace přehrály jako jedna a proto se hodí vytvořit z nich montage.

Dalším využitím montage je práce s root motion. Ten bychom využili například v předchozím případě, kdy by animace poskytovala data o posunu postavy nahoru na překážku. Montage nám tak umožňuje tento pohyb sloučit do jednoho celku.

9 Dialogový systém

NPC musí své rozhodnutí správně prezentovat. Chytré NPC je nám k ničemu, pokud hráč nepochopí jeho chování. Jeden z nejzákladnější způsobů jakým komunikovat s hráčem je pomocí dialogů. Ať už se jedná o textové dialogy nebo audio nahrávky, můžeme tímto způsobem informovat hráče o tom, co NPC dělá nebo co se chystá udělat. Navíc lze dialogy použít k předání pomocné informace hráčovi. Rozhovorem mezi několika NPC můžeme sdělit hráči informaci o jejich nepozornosti.

Při tvorbě dialogového systému je třeba se zamyslet nad typem hry, kterou vytváříme. Dialogový systém v logické adventuře bude jiný než třeba v akční hře nebo RPG. V některých případech se bude přiklánět spíše k více scénářem ovlivněným dialogům. V jiných se zase bude jednat o systémové spouštění dialogů. Pro tuto práci byl vytvořen systém, který využívá oba přístupy a dokonce je může plynule kombinovat.

Způsob implementace dialogového systému záleží na množství dialogů použitých ve hře. Pokud se bude jednat pouze o pár hlášek, které bude NPC vyvolávat v určitých situacích, můžeme dialogy nějakým způsobem uložit přímo v enginu (například pomocí datové tabulky). V našem případě budeme využívat složitější dialogový systém, který bude navíc obsahovat velké množství dialogů, které budou muset být nějakým způsobem organizovány. V takovém případě je dobré pro organizaci dialogů použít externí soubor. V našem případě se jedná o XML soubor. Můžeme si tak definovat hierarchii XML souboru, který bude mít uloženy jednotlivé dialogy i se všemi potřebnými informacemi.

Dialogový systém implementovaný v této práci je jedním z příkladů, kdy nelze využít pouze BP skripty. Součástí tohoto systému je totiž práce s XML souborem a BP neobsahují funkcionalitu pro práci s XML soubory. A právě zde můžeme využít jednu z předností Unreal enginu a naimplementovat tento systém pomocí c++. UE již obsahuje knihovnu pro práci s XML a proto ji můžeme využít pro načítáni XML souboru s dialogy. Ovšem nic nám nebrání využít jinou knihovnu pro práci s XML v c++.

9.1 Rozdělení dialogů

Dialogy jsou rozděleny do dvou kategorií: state, script. První kategorie obsahuje dialogy závislé na stavu NPC. V XML souboru můžeme vidět bloky odpovídající jednotlivým stavům. Stejným způsobem jsou tyto dialogy uloženy v dialogové komponentě. Tyto dialogy jsou spouštěny systémově například v behavior tree nebo v některém z podpůrných blueprintů. Dialogy závislé na stavu NPC mají větší prioritu než dialogy z kategorie script. Pokud tedy probíhá script dialog, bude state dialogem přerušen.

Druhou kategorií je script. Zde se nacházejí dialogy více podobné scénáři. Jedná se spíše o delší konverzace, které jsou ve většině případů manuálně spouštěny. Může se jednat o konverzaci, která posune dál příběh hry nebo například o výklad virtuálního průvodce.

```
<conversation id="2" once="0" agents="1">
   <dialogues>
      <dialogueset>
         <dialogue audio="a1" delay="1.0" duration="4.0" agentID="0"
             question="1" emotion="1">
            <text>Tohle je {0}</text>
            <distance>500.0</distance>
            <param>1</param>
         </dialogue>
         <dialogue audio="a2" delay="1.0" duration="4.0" agentID="0"
             question="1" emotion="1">
            <text>Tam daleko ta vec je {0}</text>
            <distance>1500.0</distance>
            <param>1</param>
         </dialogue>
         <dialogue audio="a3" delay="1.0" duration="4.0" agentID="0"
             question="1" emotion="1">
            <text>Je to strasne daleko, ale vypada to jako {0}</text>
            <param>1</param>
         </dialogue>
         <dialogue audio="a4" delay="1.0" duration="4.0" agentID="0"
             question="1" emotion="1">
            <text>Tak daleko nevidim.</text>
         </dialogue>
      </dialogueset>
   </dialogues>
   <fallbacks>
   </fallbacks>
</conversation>
```

Výpis 1: Ukázka dialogů s podmínkou.

9.1.1 Blok dialogue

Blok dialogue obsahuje veškeré informace k provedení dialogu. Kromě samotného textu obsahuje i název audio souboru, identifikační číslo NPC nebo číslo určující mimiku NPC. Zároveň obsahuje i data pro provádění podmínek nad dialogy. Tento dialogový systém totiž obsahuje několik jednoduchých, ale často používaných podmínek spojených s prováděním dialogů. Můžeme tak vytvářet dialogy, které se mění v závislosti na vzdálenosti od pozorovaného objektu nebo například počtu parametrů. A právě parametry jsou dalším obohacením tohoto systému.

Jedná se funkcionalitu použitelnou spíše v textové podobě, kdy můžeme na místo parametru dynamicky vložit text. Můžeme tedy přiřadit některým objektům ve scéně jména a tyto jména pak použít, jakmile bude NPC o daném objektu hovořit.

9.1.2 Blok dialogueset

Tento blok může obsahovat jeden a více dialogů. Je používán jako seřazený seznam dialogů. Primárně se používá pro výběr dialogů s podmínkami. Dialogy se testují podle toho, jak jsou seřazeny a jakmile některý z nich splňuje podmínku, ostatní se již testovat nebudou. Znamená to, že bez ohledu na to, kolik dialogů dialogset obsahuje, vždy se z něj vybere pouze jeden a to ten, který jako první splní všechny své podmínky. Navíc nám dialogueset dovoluje vložit na konec dialog bez podmínek, který bude vyvolán, pokud všechny ostatní selžou.

9.1.3 Blok conversation

Tento blok obsahuje informace o tom, kolik NPC se konverzace účastní, zda může být konverzace provedena více než jednou nebo zda musí mít všechny NPC stejný stav pro spuštění této konverzace. Uvnitř tohoto bloku najdeme další dva bloky. Blok dialogues a fallback obsahují dialoguesety. Rozdíl je v tom, že blok dialogues obsahuje primární dialogy, které se budou provádět. Blok fallback slouží jako úložiště pro dialogy, které se vyvolají při přerušení jinou konverzací. Přesněji řečeno po skončení dialogu, který přerušil aktuální dialog.

9.2 Inicializace dialogů

Předtím než se spustí konverzace mezi NPC, je třeba tuto konverzaci inicializovat. NPC může vést konverzaci sám nebo s ostatními NPC. Pokud se konverzace účastní více NPC, je třeba vybrat vedoucího konverzace (dále master). Ten má na starost inicializaci a běh konverzace. Během inicializace musí předat všem ostatním NPC účastnícím se konverzace data o konverzaci a nastavit výchozí hodnoty konverzace. Master navíc obsahuje reference na ostatní účastníky konverzace. Pokud je inicializovaná konverzace závislá na stavu NPC, přeruší aktuální konverzaci a spustí se místo ní. Konverzace závislé na stavu NPC totiž mají vyšší prioritu. Při inicializaci je možné určit, zda chceme aby konverzace začala ihned po inicializaci nebo zda si přejeme konverzaci spustit manuálně. Inicializovat můžeme konverzaci buď podle identifikačního čísla, které určí konverzaci z XML souboru a nebo můžeme konverzaci vložit manuálně. Manuální způsob se hodí například pro krátké konverzace spouštěné konkrétními objekty v herní úrovni.

9.3 Provedení dialogů

Za chod konverzace je zodpovědné NPC, které konverzaci inicializovalo. Po inicializaci se prochází konverzace po jednotlivých dialoguesetech. Z těch se vybere adekvátní dialog a předá se konkrétnímu NPC 17. To následně dialog provede a zapne časovač, po kterém se dialog ukončí. Délka dialogu je dána délkou audio souboru přiřazeného ke konkrétnímu dialogu. Pokud

Obrázek 17: Konverzace mezi dvěma NPC

dialog neobsahuje audio soubor, použije se délka určená v XML souboru s dialogy. Jakmile NPC ukončí dialog, vrátí veškeré potřebné proměnné do původní podoby a zavolá na master NPC pokračování dialogu.

10 Implementace ukázkové aplikace

V ukázkové aplikaci jsou demonstrovány postupy popsány v této práci. První část této aplikace se zabývá UI pomocníků. Demonstruje pohyb UI pomocníka a jeho interakci s hráčem a okolním světem. Tyto interakce jsou výsledkem jak systémových rozhodnutí UI tak i předpřipravených akcí.

Druhá část aplikace demonstruje funkčnost zraku, sluchu, koordinovaného pohybu a orientaci v prostředí. Místo UI pomocníka bylo v této části využito UI nepřátel. Díky tomu můžeme demonstrovat i funkčnost většího počtu NPC a jejich interakce se světem.

10.1 Návrh ukázkové aplikace

První část ukázkové aplikace demonstruje UI pomocníka. Jedná se o herní úroveň s pevně daným začátkem a koncem, ovšem několika různými cestami. Účelem této části ukázkové aplikace je demonstrovat pohyb UI pomocníka, jeho interakci s hráčem a prostředím. Proto bylo zapotřebí při designu úrovně rozmístit aktivní prvky tak, aby s nimi mohl pomocník případně interagovat. Pokud se tedy hráč zdrží v nějaké lokaci, UI pomocník začne tuto lokaci prozkoumávat. Body zájmu, které určují zajímavé místa pro pomocníka, mohou spustit dialog nebo například animaci. Po celé mapě jsou zároveň rozmístěny tzv. dialogové zóny. Pokud jimi pomocník projde, může se spustit jeden z přiřazených dialogů. Těmito aktivními prvky byla vytvořena variabilita i při více průchodech, jelikož akce UI a dialogy nejsou pevně určeny. UI pomocník zároveň funguje jako průvodce pro hráče. Je tak možné vidět implementaci pohybu pomocí vodícího objektu. Pokud tedy hráč půjde přibližně směrem k cíli, UI pomocník půjde před hráčem. Tímto způsobem lze navodit pocit, že UI pomocník vede hráče k cíli i přesto, že je jeho pohyb závislý na pohybu hráče.

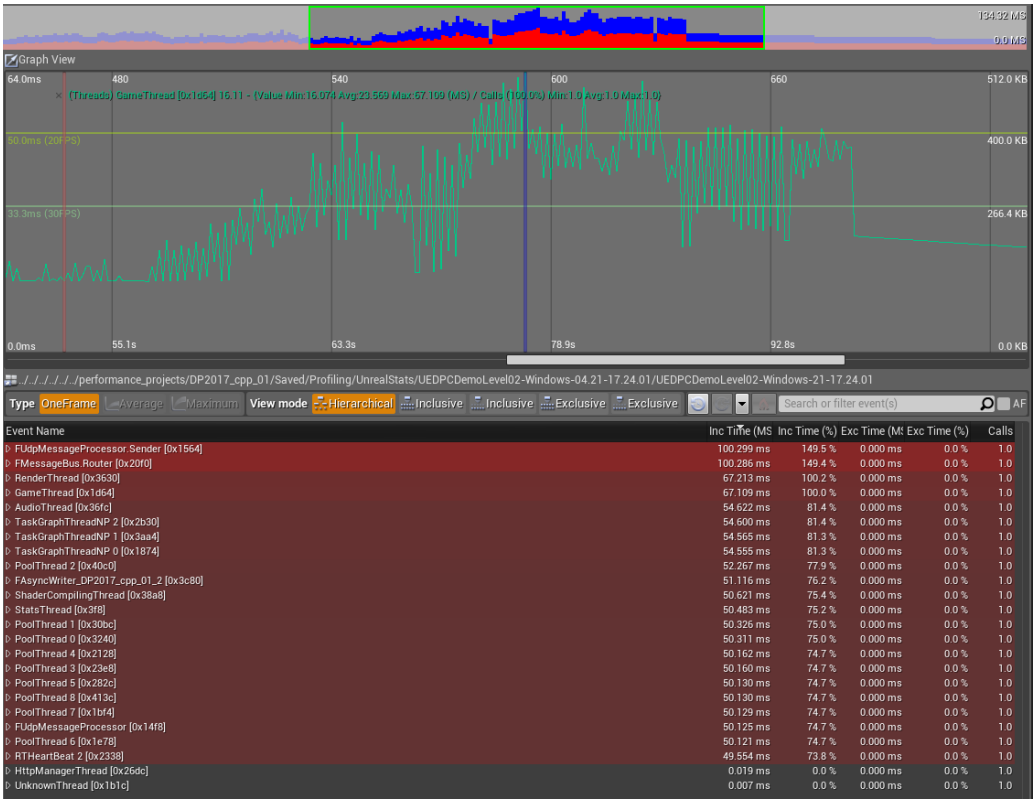
Ve druhé části ukázkové aplikace se zabýváme UI nepřátel. Pro tento účel byla vytvořená větší a otevřenější mapa, než v první části. Cílem hráče je posbírat tři speciální objekty a poté se dostat do cílové lokace. To, jaký přístup hráč zvolí, je čistě na něm. Může využít akčního postupu úrovní, ale také se může rozhodnout pro plíživý postup. Mapa je navržena pro oba způsoby hratelnosti. Nachází se tam tak spoustu úkrytů, za které se může schovat jak hráč tak i NPC. Navíc jsou úkryty rozmístěny tak, aby nebyl postup úrovní příliš jednoduchý. Kromě klasických překážek, jako jsou různé zídky a sloupy, byly po úrovni rozmístěny i úkrytové zóny. Jedná se o zónu, která hráče ukryje systémově, nikoliv zamezením viditelnosti. Tento typ úkrytů se dá použít v kombinací s modely vysoké trávy nebo například v temných zákoutích, ve kterých by NPC nemělo hráče vidět. Jelikož je nepřátelské území uprostřed mapy, může hráč vstoupit na nepřátelské území ze všech stran.

Obrázek 18: Ukázková aplikace

10.2 Optimalizace ukázkové aplikace

Během tvorby ukázkové aplikace byla provedena řada optimalizací. Oproti testovacím úrovním, které byly využity pro testování jednotlivých systémů UI, je ukázková aplikace komplexnější. Jedná se o daleko větší a členitější úroveň. Zároveň je v ukázkové aplikaci více NPC. Oproti testovacím úrovním, kde bylo většinou použito 1-5 NPC, je v ukázkové aplikace asi 30 NPC. To může způsobovat propady snímkové frekvence, které jsme u testování s malým počtem NPC nezaznamenali. V takovém případě je zapotřebí využít nástroje profiler, který nám umožňuje monitorovat náročnost jednotlivých částí hry.

V případě ukázkové aplikace se objevil problém v momentě, kdy NPC začala útočit na hráče. Propad snímkové frekvence byl v tomto případě velmi velký, kdy nejnižší hodnota dosahovala pouhých 15 snímků za sekundu. Po prozkoumání dat v profileru 19 byly objeveny dva problémy. Prvním z nich byla simulace šíření zvuku. Jelikož je úroveň ukázkové aplikace velmi členitá, šíření zvuku se ve většině případů simulovalo pomocí hledání cesty po navigační geometrii. To v případě boje s NPC, kdy hráč střílí a rychle se pohybuje, znamenalo, že velké množství NPC velmi často zpracovávalo zvuk právě tímto způsobem. Jak již bylo zmíněno v kapitole o implementaci sluchu, hledání cesty po navigační geometrii je náročná operace. V tomto případě se jednalo o řádově desítky provedení této funkce za sekundu, což mělo za následek velmi velký propad snímkové frekvence. Tento problém byl vyřešen změnou způsobu, jakým se vyhodnocuje šíření zvuku během boje. Místo tohoto způsobu byla použita jednoduchá kontrola vzdálenosti. Ta sice není tak přesná a nerespektuje geometrii herní úrovně, ovšem během boje si toho hráč prakticky nevšimne.



Obrázek 19: Profiler - graf neoptimalizované aplikace

Druhým problémem, který způsoboval propady snímkové frekvence, bylo přehlcení zvuky. Každá postava vydává při každém kroku zvuk. Navíc i každý výstřel má svůj zvuk. A tak během boje, kdy hráč může bojovat i s desítky nepřátel, se spouštělo velmi velké množství zvuků najednou. To způsobovalo jak výkonové problémy, tak i nesprávné přehrávání zvuku, který byl neustále přerušován jinými zvuky. Řešením tohoto problému bylo přiřazení jednotlivým zvukům tzv. sound concurrency. Jedná se o třídu, která má na starost omezení přehrávání zvuků. Dovoluje nám nastavit maximální počet zvuků, které se mohou najednou přehrávat, ale také metodu, jaká bude použita pro vyřazení ostatních zvuků. V našem případě jsem tak mohli omezit zvuky kroků a výstřelů na 16 a ostatní vyřadit podle vzdálenosti.

Další optimalizací, která snížila náročnost NPC, byla optimalizace animací. Náročnost animací v této aplikaci není závislá na stavu NPC, takže nezpůsobuje náhlé výkyvy snímkové frekvence. Ovšem při množství NPC, použitých v ukázkové aplikaci, mohou mít neoptimalizované animace znatelný negativní vliv. U animací je jeden ze základních způsobů optimalizace tvorba tzv. LOD. Náročnost animace je totiž závislá na počtu vertexů, které jsou ovlivněny pohybem kostry. V případě modelu robota, použitého v této aplikaci, se jedná o více než 23000 vertexů. Vzhledem k tomu, že je spousta NPC daleko od hráče nebo dokonce zakrytá jinými objekty, je zbytečné provádět animace nad tak složitými modely. Proto byly vytvořeny tři další modely tohoto robota, se sníženým počtem vertexů. Navíc pokud je hráč od NPC dostatečně

daleko, animace se vypnou úplně. Je totiž zbytečné provádět jakékoliv výpočty pro animace, pokud konkrétní NPC nevidíme nebo je tak daleko, že by to stejně nebylo poznat.

Optimalizací se dočkal i systém zraku. Implementace zraku v této práci má výhodu v tom, že odděluje detekování hráče, NPC a objektů. Díky tomuto rozdělení můžeme optimalizovat zrak jednoduchou změnou parametrů. Například nepřátelé v ukázkové aplikaci mají vypnuto detekování objektů a interval detekce ostatních NPC mají nastaveno na 0.5 s. Detekování hráče je pak nastaveno na 0.1 s. U detekování NPC by ovšem mohl nastat problém. Zrak totiž při detekování NPC plní pole referencemi na ostatní NPC které vidí. Toto pole se tak při každé detekci vyprázdní a znova zaplní. Mohl by tak nastat případ, kdy se potká velké množství NPC a mazání a realokace pole by mohlo představovat problém. Proto byl systém zraku upraven tak, aby byla paměť pro pole detekovaných NPC dopředu alokovaná. Navíc bylo místo funkce pro mazání pole empty, použitá funkce reset, která neuvolňuje paměť pole.

11 Závěr

Tato práce se zabývá herní umělou inteligencí a jednotlivými částmi, které ji tvoří. V rámci této práce byly stručně popsány jednotlivé typy umělé inteligence. Dále byl popsán aktuální stav herní umělé inteligence, přičemž se popis soustředí na humanoidní umělou inteligenci. Právě humanoidní umělá inteligence je hlavní náplní této práce.

Dále tato práce obsahuje podrobný popis systému vnímání umělé inteligence. Ten se soustředí na různé varianty systému vnímání a jejich výhody a nevýhody. Zároveň byl naimplementován vlastní systém vnímání a proces této implementace podrobně popsán. Vzorem pro implementaci zraku a sluchu byly moderní hry využívající podobné systémy, přičemž rozdíly v implementaci byly odůvodněny a podrobně popsány.

Pro implementaci rozhodování umělé inteligence byl v této práci použit systém behavior tree. Ten byl využit při tvorbě UI pomocníka a nepřátel. Vnitřní struktura obou typů UI a jejich implementace byla podrobně popsána. Zároveň se práce zabývá i ostatními podpůrnými systémy pro efektivnější rozhodování UI.

Umělá inteligence byla rozšířená pomocní animačního systému, který nám dovolil měnit animace v závislosti na stavu UI. Pro zpracování animací byl využit animační systém Unreal enginu. Při implementaci UI byly animace využity nejen jako vizuální část, ale také jako funkční prvek přímo ovlivňující pohyb a tudíž i akce UI. Zároveň byl v této práci naimplementován dialogový systém, umožňující konverzaci několika NPC. Tento systém využívá textové i audio dialogy.

Pro demonstraci postupů popsaných v této práci byla vytvořená ukázková aplikace. Ta se zabývá umělou inteligencí pomocníků a nepřátel. Během implementace ukázky UI pomocníka byl kladen důraz na interakci UI s okolním prostředím. Zde se osvědčila kombinace systémových a předpřipravených akcí. Při implementaci ukázky s UI nepřáteli byl kladen důraz na demonstraci systému vnímání, ale také koordinovaného pohybu a využití prostředí.

V této práci byla využita jen velmi omezená sada animací a pro tvorbu dialogů byl využit generátor řeči. Jelikož je prezentace velkou částí umělé inteligence, dala by se tato práce rozšířit právě o kvalitnější animace a dialogy. Dalším možným rozšířením by byla implementace spolupráce dvou a více UI pomocníků. Zde by byl potenciál pro ještě živější a přirozenější interakce UI jak s herním světem tak i mezi sebou.

Literatura

[1] *Mass Effect 1*[online], Electronic Arts Inc., 2007, <http://www2.ea.com/mass-effect-1>

[2] *Mass Effect: Andromeda*[online], Electronic Arts Inc., 2017, <https://www.masseffect.com/>

[3] *Maybe We Can't Handle Smart Enemies In Our Games*[online], Patricia Hernandez, Gizmodo Media Group, 2014, <http://kotaku.com/maybe-we-cant-handle-smart-enemies-in-our-games-1542300804>

[4] *Age of Empires II*[online], Microsoft Studios, 1999, <https://www.ageofempires.com/>

[5] *Stronghold*[online], Firefly Studios, 2001, <http://fireflyworlds.com/games/stronghold/>

[6] *Heroes of Might and Magic*[online], Ubisof Entertainment, 2015, <https://mmh7.ubi.com/en/blog>

[7] *Left 4 Dead*[online], Valve Corporation, 2009, <http://www.valvesoftware.com/games/l4d2.html>

[8] *Shadow Tactics: Blades of the Shogun*[online], Daedalic Entertainment GmbH, 2016, <http://www.shadow-tactics.com/>

[9] RABIN, Steve, *Game AI pro: collected wisdom of game AI professionals*, A K Peters/CRC Press, 2013, 626 s. ISBN 978-1466565968

[10] RABIN, Steve, *Game AI pro 2: collected wisdom of game AI professionals*, A K Peters/CRC Press, 2015, 577 s. ISBN 978-1482254792

[11] *The Last of Us*[online], Sony Interactive Entertainment America LLC, 2013, <https://www.naughtydog.com/games/the_last_of_us/>

[12] *Splinter Cell: Blacklist*[online], Ubisoft Entertainment, 2013, <https://www.ubisoft.com/en-US/game/splinter-cell-blacklist/>

[13] *Recast*[online], Mikko Mononen, 2013, <http://masagroup.github.io/recastdetour/index.html>

[14] MARK, Dave, *Behavioral mathematics for game AI*, Boston, MA: Charles River Media, Course Technology, Cengage Learning, c2009, 480 s. ISBN 978-1584506843

[15] *Ellie: Buddy AI in The Last of Us*[online], Max Dyckhoff, 2014, <http://gdcvault.com/play/1020364/Ellie-Buddy-AI-in-The>

[16] *The Last of Us: Human Enemy AI*[online], Travis McIntosh, 2014, <http://gdcvault.com/play/1020338/The-Last-of-Us-Human>

[17] *Unreal Engine wiki*[online], Epic Games, Inc, 2017, <https://wiki.unrealengine.com/>

[18] *Unreal Engine documentation*[online], Epic Games, Inc, 2017,<https://docs.unrealengine.com>

A Příloha na DVD

Součástí diplomové práce je DVD obsahující ukázkovou aplikaci, projekt se zdrojovými kódy, soubory s dialogy a elektronickou verzi této práce.

Obsah DVD:

- \Aplikace\wlo0008.exe
- \Projekt\wlo0008.uproject
- \Dialogy
- \Diplomova_prace_Jan_Wlosok.pdf
- \README.txt